Holistic Security Engineering for Software-Defined Road Vehicles

by

Dominik Püllen

A

Dissertation

submitted to the
Faculty of Computer Science and Mathematics
at the
University of Passau

In Partial Fulfilment of the Requirements
for the Degree of
Doctor of Engineering (Dr.-Ing.)

1st Reviewer:	Prof. Dr. Stefan Katzenbeisser
2nd Reviewer:	Prof. Dr. Frank Kargl
Day of Submission:	22.02.2024
Day of Oral Examination:	19.06.2024

Passau 2024

Declaration of Originality

I, Dominik Püllen, hereby declare that this Ph.D. thesis, titled "Holistic Security Engineering for Software-Defined Road Vehicles", is entirely my own work, except where otherwise indicated. All ideas, research findings, analysis, and conclusions presented in this thesis are the result of my own independent research and intellectual effort.

I acknowledge and understand the ethical and academic responsibilities associated with producing original research. I have appropriately cited and referenced all sources of information, including but not limited to:

- Any published or unpublished works, including books, articles, reports, and electronic sources, that have been directly quoted, paraphrased, or summarized in this thesis.
- Any assistance or contributions from fellow researchers, advisors, or individuals who have provided guidance, data, or other forms of support for this research.
- Any collaborative work or contributions from co-authors or collaborators in publications or projects related to this research.

I have complied with all relevant ethical standards and academic conventions regarding the proper citation and acknowledgment of sources. I have used automated tools such as Grammarly to correct orthographic and grammatical mistakes.

Dominik Püllen	Date

Abstract

With the increasing use of digital technologies in the automotive sector, the traditional automobile is undergoing a structural transformation, requiring new technologies and enabling innovative mobility concepts. In particular, the ability to drive automatically or even fully autonomously, update control software, and remain connected to the environment allows attackers to infiltrate highly critical vehicle systems and take control without adequate protection. Once not only individual vehicles but entire fleets are dominated by software, cyberattacks could disrupt a significant portion of the infrastructure and expose passengers to substantial risks.

This work follows a holistic approach to protecting highly automated software-defined vehicles from cyberattacks by designing and implementing security concepts in the main phases of a vehicle's lifecycle. We use SAE level 4 prototype vehicles to evaluate our proposed techniques. We start with a systematic security requirement analysis using the ISA-62443 standard series, demonstrating how threats can be identified in a collaborative, hierarchical process and how the resulting security risks impact the software and hardware architecture of a self-driving vehicle. We show how this analysis process results in concrete requirements whose consideration reduces the overall security risk to a tolerable level.

Subsequently, we develop technical solutions for selected requirements. We begin by securing the Controler Area Network (CAN) and FlexRay legacy protocols, which we foresee being used in specific areas of Software-Defined Vehicles (SDVs) in a transitional period despite technological changes. To enable vehicle-wide security management, we address the management and distribution of cryptographic keys within such networks, mainly focusing on resource-constrained devices. We propose using lightweight implicit certificates for deriving cryptographic group keys that can be used in CAN networks. Additionally, we demonstrate how the slot-based frame structure of the FlexRay protocol allows for efficient "multi-slot" authentication, for which we calculate cryptographic keys using hash-based key chains.

SDVs use Ethernet-based communication protocols and custom middleware stacks to transmit large amounts of data in real-time. We develop a three-stage security process for the novel Automotive Service-Oriented Architecture (ASOA), which enables the development and central orchestration of system-agnostic functional software components on embedded systems and High Performance Computing (HPC) platforms. After the central specification of the security architecture at the data flow level, security tokens are automatically calculated and distributed for runtime protection of the service-oriented, DDS-based data transmission. Our process ensures the strict separation of function and system knowledge, allowing for cost-effective and adaptable security architecture management. The evaluation in four self-driving, software-defined vehicles demonstrates an average runtime overhead of approximately 5.71%.

As the initial risk analysis and actual cyberattacks have shown, protective measures against the compromise of control units must be taken alongside communication security. To address this, we develop a method for verifying and validating the software integrity of control units. A governmental third party confirms a measurement through a digital certificate, proving the examined vehicle's trustworthiness and suitability for participation in automated traffic.

In the final step of this work, we present an assessment scheme that allows software-defined vehicles to evaluate security incidents during operation in terms of their maximum expected damage and initiate appropriate countermeasures. We follow the ISO/SAE 21434 standard and model attack paths using a graph representing dependencies among internal vehicle assets to account for the propagation effects of cyberattacks. The assessment of a security incident considers not only the probability of individual attack paths but also the vehicle context. Our practical evaluation demonstrates that we can detect, report, and assess security incidents below the human reaction time in the earlier mentioned prototype vehicles.

Zusammenfassung

Mit dem vermehrten Einsatz digitaler Technologien im Automobilbereich erfährt das klassische Fahrzeug einen strukturellen Wandel, der neuer Technologien bedarf und innovative Mobilitätskonzepte ermöglicht. Insbesondere die Fähigkeit, automatisiert oder sogar vollständig autonom zu fahren, Steuersoftware zu aktualisieren und dabei mit dem Umfeld vernetzt zu bleiben, ermöglicht es Angreifern, in hochkritische Fahrzeugsysteme einzudringen und die Kontrolle bei ungenügendem Schutz zu übernehmen. Sobald nicht nur einzelne Fahrzeuge von Software dominiert werden, sondern ganze Flotten, könnten Hackerangriffe einen Großteil der Infrastruktur lahmlegen sowie die Passagiere großen Gefahren aussetzen.

Diese Arbeit verfolgt einen ganzheitlichen Ansatz zum Schutz hochautomatisierter, softwaredefinierter Fahrzeuge (SDVs) vor Cyberangriffen, indem Sicherheitskonzepte in den Hauptphasen des Lebenszyklus eines Fahrzeugs entworfen und umgesetzt werden. Dabei verwenden wir vier SAE-Level-4-Prototypenfahrzeuge, um unsere vorgeschlagenen Lösungen zu testen und zu bewerten. Zunächst beginnen wir mit einer systematischen Sicherheitsanforderungsanalyse unter Verwendung der ISA-62443-Normenreihe, um zu zeigen, wie Bedrohungen in einem kooperativen, hierarchischen Prozess identifiziert werden können und wie die resultierenden Sicherheitsrisiken die Software- und Hardwarearchitektur eines selbstfahrenden Fahrzeugs beeinflussen. Wir zeigen, wie dieser Analyseprozess in konkreten Sicherheitsanforderungen resultiert, dessen Berücksichtigung das Gesamtrisiko auf ein tolerierbares Niveau senkt.

Im Anschluss entwickeln wir für ausgewählte Anforderungen technische Lösungen. Wir beginnen mit der Absicherung des Controler Area Network (CAN) und des FlexRay Legacy-Protokolls, deren Einsatz wir trotz des technologischen Wandels auch in nächster Zukunft an spezifischen Stellen in software-definierten Fahrzeugen sehen. Um ein fahrzeugweites Sicherheitsmanagement zu ermöglichen, befassen wir uns mit der Verwaltung und Verteilung kryptographischer Schlüssel innerhalb solcher Netzwerke und betrachten vor allem ressourcenbeschränkte Geräte. Wir schlagen die Verwendung impliziter Zertifikate zur Ableitung von Gruppenschlüsseln in CAN-Netzwerken vor. Außerdem zeigen wir, wie die slotbasierte Framestruktur des FlexRay-Protokolls eine effiziente *multi-slot* Authentifizierung ermöglicht, für welche wir die kryptographischen Schlüssel mittels hashbasierter Schlüsselketten berechnen.

Für die Echtzeitübertragung großer Datenmengen, die beispielsweise bei der Auf- und Verarbeitung von Sensordaten entstehen, verwenden software-definierte Fahrzeuge Ethernet-basierte Übertragungsprotokolle und eigene Middleware-Stacks. Wir erarbeiten einen dreistufigen Sicherheitsprozess für die neuartige Automotive Service-Oriented Architecture (ASOA), welche die Entwicklung und zentrale Orchestrierung systemagnostischer funktionaler Softwarekomponenten auf eingebetteten Systemen sowie auf HPC-Plattformen ermöglicht. Nach der zentralen Spezifikation der Sicherheitsarchitektur auf Datenflussebene erfolgt die automatische Berechnung und Verteilung von Sicherheitstokens, welche zur Laufzeit für den Schutz der service-orientierten,

DDS-basierten Datenübertragung verwendet werden. Unser Prozess berücksichtigt die strikte Trennung von Funktion und Systemwissen und ermöglicht daher eine kostengünstige sowie adaptierbare Verwaltung der Sicherheitsarchitektur. Die Evaluation in vier selbstfahrenden, software-definierten Fahrzeugen belegt den vernachlässigbaren gemittelten Laufzeit-Overhead von rund 5.71%.

Wie die anfänglich durchgeführte Risikoanalyse sowie tatsächliche Cyberangriffe zeigen, müssen neben der Kommunikationsabsicherung Schutzmaßnahmen zur Kompromittierung von Steuergeräten ergriffen werden. Dazu entwickeln wir ein Verfahren zur Verifizierung und Validierung der Software-Integrität von Steuergeräten. Eine hoheitliche dritte Partei bestätigt eine erfolgte Integritätsmessung durch ein digitales Zertifikat, das die Vertrauenswürdigkeit des untersuchten Fahrzeuges und damit dessen Eignung zur Teilnahme am automatisierten Verkehr belegt.

Im letzten Schritt dieser Arbeit präsentieren wir ein Bewertungsschema, das es software-definierten Fahrzeugen im laufenden Betrieb ermöglicht, Sicherheitsvorfälle hinsichtlich ihres maximal zu erwartenden Schadens zu bewerten und entsprechende Gegenmaßnahmen einzuleiten. Dazu orientieren wir uns am ISO/SAE 21434 Standard und modellieren Angriffspfade anhand eines Graphen, der Abhängigkeiten fahrzeuginterner Assets repräsentiert, um Propagierungseffekte von Cyberangriffen zu berücksichtigen. In die Bewertung eines Sicherheitsvorfalls fließt neben der Wahrscheinlichkeit einzelner Angriffspfade auch der Fahrzeugkontext ein. Unsere praktische Evaluation zeigt, dass die Detektion, Meldung und Bewertung von Sicherheitsvorfällen in einem der zuvor genannten prototypischen Fahrzeuge unterhalb der menschlichen Reaktionszeit möglich ist.

List of Publications

Peer-reviewed publications used in this thesis:

[1] Dominik Püllen, Nikolaos Anagnostopoulos, Tolga Arul, Stefan Katzenbeisser. "Safety meets security: Using IEC 62443 for a highly automated road vehicle." In: *Computer Safety, Reliability, and Security: 39th International Conference*, SAFECOMP 2020, Lisbon, Portugal, September 16–18, 2020, Proceedings 39, pp. 325-340. Springer International Publishing, 2020.

[2] Dominik Püllen, Nikolaos Anagnostopoulos, Tolga Arul, Stefan Katzenbeisser. "Using implicit certification to efficiently establish authenticated group keys for in-vehicle networks." In: *IEEE Vehicular Networking Conference (VNC)*. 2019.

[3] Dominik Püllen, Nikolaos Anagnostopoulos, Tolga Arul, Stefan Katzenbeisser. "Securing FlexRay-based in-vehicle networks." In: *Microprocessors and Microsystems* 77 (2020): 103144.

[4] Dominik Püllen, Florian Frank, Marion Christl, Wuhao Liu, Stefan Katzenbeisser. "A Security Process for the Automotive Service-Oriented Software Architecture." In: *Transactions on Vehicular Technology (2023)*.

[5] Dominik Püllen, Nikolaos Anagnostopoulos, Tolga Arul, Stefan Katzenbeisser. "Poster: Hierarchical Integrity Checking in Heterogeneous Vehicular Networks." In: *IEEE Vehicular Networking Conference (VNC)*. 2018.

[6] Dominik Püllen, Felix Klement, Alexey Vinel, Stefan Katzenbeisser. "Ensuring Trustworthy Automated Road Vehicles: A Software Integrity Validation Approach." In: *IEEE International Automated Vehicle Validation Conference*, 2023

[7] Dominik Püllen, Jonas Liske, and Stefan Katzenbeisser. "ISO/SAE 21434-Based Risk Assessment of Security Incidents in Automated Road Vehicles." In: *Computer Safety, Reliability, and Security: 40th International Conference*, SAFECOMP 2021, York, UK, September 8–10, 2021, Proceedings 40. Springer International Publishing, 2021.

Further publications:

[8] Florian Kohnhäuser, Dominik Püllen, and Stefan Katzenbeisser. "Ensuring the Safe and Secure Operation of Electronic Control Units in Road Vehicles." In: *IEEE Security and Privacy Workshops (SPW)*. 2019.

[9] Timo Woopen, Bastian Lampe, ..., Dominik Püllen, et al. "UNICARagil - Disruptive Modular Architectures for Agile, Automated Vehicle Concepts", In: *27. Aachen Colloquium Automobile and Engine Technology*, 2018.

Preface

I would like to take the opportunity to express my gratitude to all those who have supported me throughout the journey of writing this dissertation. First, I extend my thanks to my supervisor, Stefan Katzenbeisser, for allowing me to conduct my research under his guidance. I appreciated much the freedom he granted me in shaping my daily work routine while continually offering the possibility for consultation. Beyond the original research, I am grateful for the management skills I acquired under his supervision. The experience of overseeing various research tasks and projects, hiring supporting personnel, and applying for project funding has broadened my horizons. In particular, I am grateful for being in charge of the UNICARagil project, which paved my academic path and significantly inspired my research. This project enabled me to work in a large interdisciplinary team of researchers and taught me to collaborate with smart people of different backgrounds. Therefore, I also express my appreciation to the German taxpayer and the Federal Ministry of Education and Research for their public funding support, which has been instrumental in the success of UNICARagil.

I also extend my thanks to Frank Kargl, who accepted the role of second examiner for this dissertation. His thorough examination and expert insights have contributed to the quality of this work.

Additionally, I thank my colleagues for our chair's supportive atmosphere and social environment. Our friendly relations made me come joyous to the office and created a motivating atmosphere, even during challenging times like the coronavirus outbreak. I learned how team spirit and a strong company positively impact personal effectiveness. Special thanks are also due to Marion Christl, whose assistance as a student researcher has been very helpful to me.

Shortly before submitting this thesis, my grandfather, Horst Graebe, passed away. He was determined to read this thesis. I would therefore like to pay special tribute to him at this point.

Lastly, and most importantly, I want to express my profound gratitude to my family, especially my wonderful wife, Marta. Relocating to Passau after our chair moved from the Technical University of Darmstadt was a decision we made together, and her presence and encouragement have been a driving force behind my perseverance and determination to complete this thesis.

To all those mentioned above and to everybody who played a part in shaping this dissertation, thank you very much!

Passau, February 2024

To Our Lady Mary

Table of Contents

List of Figures

List of Tables

List of Acronyms

ABS Anti-lock Braking System

ADG Asset Dependency Graph

AEAD Authenticated Encryption with Associated Data

AHP Analytic Hierarchy Process

ASOA Automotive Service-Oriented Architecture

ASP ASOA Security Platform

AUTOSAR AUTomotive Open Systems ARchitecture

CA Certificate Authority

CAL Cybersecurity Assurance Level

CAN Controler Area Network

CAN-FD CAN with Flexible Data rate

DDS Data Distribution Service

DoS Denial-of-Service

DTLS Datagram Transport Layer Security

E/E Electrical/Electronic

ECC Elliptic Curve Cryptography

ECDH Elliptic Curve Diffie-Hellman

ECQV Elliptic Curve Qu-Vanstone

ECU Electronic Control Unit

FPGA Field Programmable Gate Array

FSBL First-Stage Bootloader

HMAC Hash-based Message Authentication Code

HMI Human-Machine Interface

HPC High Performance Computing

HSM Hardware Security Module

IACS Industrial and Automation Control System

IMA Integrity Management Architecture

KDF Key Derivation Function

LIN Local Interconnect Network

MAC Message Authentication Code

MITM Man-In-The-Middle

MOST Media Oriented Systems Transport

NIST National Institute of Standards and Technology

NIT Network Idle Time

OMG Object Management Group

OP-TEE Open Portable Trusted Execution Environment

OS Operating System

OTP One-Time Pad

PKI Public Key Infrastructure

PUF Physical Unclonable Function

QoS Quality of Service

RoA Range of Awareness

ROS Robot Operating System

RPi Raspberry Pi

RPU Runtime Protection Unit

RTPS Real-Time Publish-Subscribe

RU Registration Unit

SDV Software-Defined Vehicle

SGX Software Guard Extensions

SM Secure Monitor

SOME/IP Scalable Service-Oriented Middleware over IP

SUC System Under Consideration

TA Trusted Application

TARA Threat Analysis and Risk Assessment

TCB Trusted Computing Base

TEE Trusted Execution Environment

TSU Trusted Software Authority

ZCR Zone and Conduit Requirement

1. Introduction

From time immemorial, mobility has been an intrinsic part of humankind, playing a crucial role in shaping societies, economies, and cultures. The advancement in transportation methods has always been crucial to economic growth and wealth, allowing humans to bridge increasingly large distances and convey goods around the globe. History is marked by several milestones that significantly contributed to the efficient, affordable, widely available, and most recently, climate-friendly locomotion: The invention of the wheel in antiquity, the utilization of coaches in medieval times, the advent of the steam engine during the 18th century, the development of the first combustion engine in the late 19th century, and ultimately, the mass production of road vehicles in the 20th century are turning points that massively boosted society in terms of technological progress and wealth. Nowadays, road vehicles have become indispensable to our everyday lives, in particular, because automobiles can be privately owned and customized.

The production of road vehicles entails a large industry involving numerous companies as suppliers. They all contribute to the growth and power of today's economies, especially in Germany, where many world-leading manufacturers along the entire automotive supply chain are located. Consequently, the automotive industry is by far the strongest industrial sector in Germany regarding turnover and exports [10], and hence, has significantly contributed to the country's prosperity. Specifically, Germany is strong in traditional mechanical engineering, which, over decades, has been a synonym for the technological innovation and competitive advantages of road vehicles, e.g., through more efficient and powerful combustion engines or increased safety and comfort at high quality.

With the mass production of vehicles beginning in the early 20th century, the aspect of safety increasingly attracted attention, and awareness for protecting the driver and passengers began rising. Contemporary road vehicles are equipped with numerous life-protecting measures such as airbags, distance control systems, and a Anti-lock Braking System (ABS). Today, constructing and deploying road vehicles is subject to regulation and certification in most jurisdictions, although basic safety means like seat belts only became mandatory in Germany in 1979.

The complexity of producing safe and reliable automobiles led to a complex development process that includes stages for planning, engineering, prototyping, testing, quality control, mass production, and long-term support. Until today, this cycle is rather stiff and time-consuming. On average, it takes a manufacturer four to five years from designing to producing a vehicle [11]. Many stakeholders are involved in a vehicle's development process and must consider technological and legal requirements. Eventually, manufactured automobiles contain many driving features and safety measures, yet their internals have traditionally remained hidden from external parties, rendering them effectively a closed system.

1.1 The Vision of Autonomous Driving

In the last decade, further technological turning points have started disrupting the automotive industry, similar to those mentioned above:

First, the electrification of road vehicles makes established propulsion methods superfluous, such that the industry must make significant investments in this technology for the sake of its survival. The transition to electric propulsion methods is a prominent political goal for environmental reasons because the required electricity can originate from climate-friendly, renewable sources. These considerations have become so significant that the European Union intends to phase out combustion engines by 2035 [12] despite their long-standing role as a driving economic force. Since the underlying technology fundamentally differs from traditional combustion engines, new market entrants compete with the established industry, making the future of the automotive market unclear.

Second, the continuous expansion of digital technologies into our daily lives not only holds the potential for transformative impacts on our society but also significantly contributes to an ongoing technological transition within the automotive domain. This transition will eventually turn road vehicles, originally mechanical devices, into software-defined computer systems on wheels.

The emergence of the Internet of Things and technologies such as artificial intelligence and cloud computing enables novel mobility ecosystems where smart, self-driving, and interconnected automobiles dominate the roads. Such SDVs illustrate how the role of software engineering becomes increasingly relevant, shifting the automotive industry's focus from traditional machine engineering to software engineering. With a growing number of automation features in contemporary road vehicles, the pace of this technological transition is picking up momentum. Envisioning a lasting future of autonomous traffic, today, the Society of Automotive Engineers distinguishes between five levels of driving automation, with humans relinquishing control to the vehicle from level three. Despite open ethical and liability questions, the automation of road vehicles encompasses clear advantages:

1. **Safety:** In 2018, the U.S. Department of Transportation stated that drivers are accountable for as much as 94% of all accidents, with vehicle defects contributing a mere 2% [13]. Consequently, the majority of vehicle accidents can be linked to human errors such as distraction, stress, missing safety distances, or speeding [14]. In contrast, automated vehicles neither get tired nor break traffic laws, thus improving passengers' safety.

2. **Driving Behavior:** Automated and interconnected traffic can positively influence traffic flows, particularly when the infrastructure is appropriately adjusted (e.g., allocating more space for drop-off locations) [15]. Context-aware driving and automatic route optimizations can result in less congestion and fewer traffic jams.

3. **Resource Utilization:** Self-driving vehicles are anticipated to operate more efficiently than their human counterparts, resulting in energy savings ranging from 11% to 55% [16]. Moreover, human passengers can utilize travel time for other productive activities.

4. **Mobility Concept:** Automated traffic facilitates the idea of on-demand driving, potentially leading to a decreased demand for parking spaces, particularly in densely populated areas. This shift opens the door for entirely new mobility concepts [17].

While improved safety is one of the proponents' strongest arguments, critics argue the opposite. Computer-based safety-critical systems, such as SDVs, possess a significantly larger attack surface, mainly due to their ability to communicate with external parties and continuously receive software updates. Unlike the traditional automobile, which resembles a closed system, SDVs are open systems, making them susceptible to cyberattacks. This concern became a reality for the first time in 2015 when researchers demonstrated how an adversary could take complete control of an off-the-shelf automobile [18], thus enormously endangering the passenger's safety. Since then, cybersecurity has moved into the focus of attention because the safety of SDVs requires the ability to detect and defend against any malicious party. In short, security is necessary to guarantee safety.

1.2 Vehicle Architectures in Transition

The increasing role of software in road vehicles due to automation and interconnection calls for innovative middleware stacks and tailored automotive operating systems, allowing for the dynamic, safe, and secure integration and maintenance of automotive functions. Historically, road vehicles have been designed as closed and rather static systems, almost resembling a black box from an engineer's perspective. This design obviously contradicts the requirements of a dynamic software architecture since deploying updates, patches, and functional upgrades becomes challenging. Besides, it is difficult, if possible at all, to effectively protect such systems from cyberattacks if security features are added in the aftermath. Therefore, not only software architectures must evolve according to security requirements, but the underlying Electrical/Electronic (E/E) architecture must also be designed in a security-aware fashion. Thus, the ongoing transition to SDVs necessitates a profound transformation of the internal architectures of these vehicles.

Conventional Domain-Based Architectures

Classical road vehicles consist of Electronic Control Units (ECUs), with each ECU typically responsible for a specific task, such as the Engine Control Module, the Transmission Control Module, the Airbag Control Module, the Tire Pressure Monitoring System, and the infotainment system. These ECUs are organized into domains, forming a domain-based E/E architecture. Within such a system, an ECU is typically an embedded system that is highly optimized regarding its electric usage, memory consumption, and processor requirements to carry out a dedicated task efficiently.

With the increasing number of vehicle functions, the quantity of ECUs has steadily grown such that modern automobiles can contain up to 150 ECUs. Traditionally, these ECUs are arranged in a bus topology, communicating through tailored, mostly signal-based automotive bus systems designed for a specific purpose. The term "signal-based" indicates that these protocols primarily exchange individual signals or data points instead of entire messages or packets. Most prominently, the CAN enables reliable data exchange between ECUs using a priority-based arbitration mechanism such that critical data is always transmitted first. When first introduced, the CAN protocol contributed significantly to reducing the wire harness and thus to cost savings. Local Interconnect Network (LIN), simpler and slower, finds use in less critical applications like mirror control. FlexRay ensures real-time communication for safety-critical ECUs with predefined time slots, while Media Oriented Systems Transport (MOST) is tailored for multimedia and infotainment systems. AUTomotive Open Systems ARchitecture (AUTOSAR) Classic standard-

izes communication interfaces and software development for the mentioned protocols in classical automotive systems.

These automotive signal-based protocols share several similarities. Firstly, they often require complex setups. For example, setting up a CAN network involves defining a communication matrix, specifying what ECU sends and receives messages of what type. In comparison, FlexRay networks require an even more complex setup, including a predefined communication schedule that assigns time slots to ECUs. Once configured, these networks typically remain unchanged and inflexible towards later modifications. Secondly, most signal-based protocols, ranging from 20 Kbit/s (LIN) to 10 Mbit/s (FlexRay), offer relatively low bandwidth, insufficient for the data transmission demands of autonomous driving and connected vehicle services. Gigabit connections are essential for real-time sensor data and environmental models within a vehicle. Thirdly, signal-based protocols inherently lack security support, posing challenges in the face of cyberattacks on modern automobiles. Although AUTOSAR SecOC [19] offers guidelines for securing signal-based communication, such solutions must be considered in an overall security process of SDVs.

Novel Zone-Based Architectures

SDVs pose technological requirements that traditional E/E architectures cannot meet. The secure and reliable transmission of potentially large amounts of real-time data is unfeasible with signal-based protocols. Furthermore, achieving the dynamic execution of software with agile development cycles while maintaining connections to external vehicle entities is impossible on conventional ECUs. The traditional approach of implementing automotive functions on dedicated ECUs becomes impractical with a continuously growing number of functions. Therefore, SDVs implement novel E/E architectures that consolidate multiple functions into centralized zones within the vehicle. These "zonal architectures" deploy fewer but more powerful, often general-purpose ECUs that can run high-performance applications as required by automated driving functions. The purpose is to ease maintainability and to reduce costs. A zone encompasses several physically proximate ECUs which are not necessarily related concerning the functions they execute. Unlike conventional architectures, where ECUs are primarily organized in domains, these zonal architectures enhance modularity and scalability, resulting in better adaptability and improved integration and deployment processes. As a result, the overall architecture complexity decreases, which is favorable for maintenance and cost reduction.

To ensure high bandwidths at low latency for data-intensive tasks, SDVs deploy Ethernet networks in a switched topology to support dynamic IP-based software architectures. Automotive software communicates using service-oriented protocols that improve scalability, interoperability, and modularity compared to signal-based protocols. While AUTOSAR Classic focuses on the development of safety-critical applications for traditional, resource-constrained ECUs, the AUTOSAR Adaptive framework reacts to the trend toward zonal architectures and service-oriented communication within vehicles by providing standardized guidelines to develop software for such environment.

Nevertheless, AUTOSAR Adaptive often faces criticism from automotive experts as it does not adequately address all demands, e.g., when it comes to the certification of safety-critical applications. As a result, OEMs, Tier 1 suppliers, and emerging companies are working on new solutions and promising automotive middleware stacks to develop safe mobility software efficiently.

For instance, the Volkswagen Group founded its subsidiary, Cariad, to develop vwOS. The Robert Bosch company and its subsidiary, the ETAS Group, are working on the Automotive Operating

System, a specific middleware solution for Advanced Driving Assistance System. Meanwhile, the recently founded company Qorix, in which the ZF Group holds a fifty-percent stake, is developing another middleware stack for modern automotive systems. APEX.AI gained much attention after receiving safety certification according to ISO 26262 for a custom ROS2 core, aiming to penetrate the automotive market with an alternative Operating System (OS). Besides commercial solutions, academic projects like the Automotive Service-Oriented Architecture (ASOA) [20] focus on developing cost-efficient, secure, and safe automotive architectures. However, it remains unclear what framework will ultimately prevail and whether the automotive industry will converge toward one or multiple solutions.

These recent developments demonstrate the tremendous market potential. According to a McKinsey report on the future of autonomous driving released in January 2023, it is projected that "by 2035, autonomous driving could generate a revenue ranging from $300 billion to $400 billion" [21].

1.3 Automotive Security

The transformation from conventional vehicles to SDVs, the associated shift from classical domain-based to zonal E/E architectures, and the deployment of dynamic software architectures constitute a protracted process within the automotive industry. It would be foolish and impossible to quickly replace well-established and reliably functioning legacy systems with emerging technologies. For example, the automotive industry has long-term contracts with suppliers and customers, making it challenging to reinvent road vehicles in a relatively short timeframe. Furthermore, vehicle manufacturers are known for their risk-averse nature, partly due to the high safety standards they have to guarantee. Since legacy systems are proven in terms of reliability and safety, they must not be demonized, but their co-existence with novel technologies is the most likely outcome. The transformation to SDVs will likely happen gradually, so modern software architectures must remain compatible with established legacy.

The acceptance of highly automated driving, which involves replacing established legacy systems with emerging technologies, depends on the assurance that these vehicles operate securely and safely. However, a 2020 Forbes report on automotive cybersecurity hacks, citing the CEO of Awen Collective, states that "nearly every manufacturer has been hacked" [22]. At the same time, the 2016 KPMG Loss Baromoter [23] revealed that 72% of consumers would hesitate to purchase from a vehicle brand that had experienced a cyberattack, while even 10% of consumers would refrain from buying from an affected brand at all. Therefore, ensuring security becomes mandatory for every automaker, particularly with SDVs possessing a larger attack surface than conventional road vehicles due to their interconnectivity and modularity.

Road vehicle security involves adjusting existing development processes because it is not solely about equipping an existing system with defense measures but rather about sensitizing the entire process with security awareness. In the interest of safety, modern vehicles must incorporate security considerations from the early phases of the development cycle, treating security as a process rather than a mere set of features. As legacy and emerging technologies will continue co-existing in the following years, both must be protected against cyberattacks and considered in a holistic security process.

1.4 Research Questions

The importance of security in road vehicles and specifically for interconnected SDVs has led to five fundamental research questions (referenced as RQ1 - RQ5) on which this dissertation is based. While RQ1 investigates the security awareness in the early design phase of a vehicle, RQ2, RQ3, and RQ4 explore defense techniques against attackers who may have control over the network or the software running on ECUs. Lastly, RQ5 focuses on the real-time monitoring and continuous assessment of the vehicle's secure state during its operational phase.

RQ1 What security requirements must SDVs meet to ward off cyberattacks?

To develop a secure and safe SDV, security awareness is essential from the very beginning. To what extent are the well-established ISA-62443 industry standards applicable to the automotive domain? In Chapter 4, we analyze and conduct each step of ISA-62443-3-2 risk analysis for a prototype vehicle platform. In this context, we explore how to model and assess threats effectively and derive security requirements.

RQ2 How can legacy signal-based protocols be safeguarded against manipulation?

How can we efficiently create and distribute authenticated cryptographic keys for CAN messages in a network with resource-constrained ECUs? Can the typical timeslot-based structure of FlexRay frames be leveraged for key computation, and is the dual-channel mode suitable for establishing a dedicated cryptography channel? We address these questions in Chapter 5.

RQ3 What does an effective security process for automotive service-oriented software middleware look like?

In Chapter 6, we examine how an emerging automotive middleware stack can be extended with an effective security process while segregating system knowledge from functions on a service level. Such a process specifies how security parameters can be configured and how identities and cryptographic keys can be managed while accommodating software updates and relocations to others ECUs. How is it possible to maintain transparency for functional developers while guaranteeing communication security between software modules?

RQ4 How can the trustworthy software state of customizable SDVs be proven?

Since the adversaries not only manipulate in-vehicle networks but also software running on ECUs, we investigate in Chapter 7 how to proactively determine the trustworthiness of automotive software while allowing for the customization of specific uncritical software components. How can an SDV prove the absence of compromised software to a third party, such as a legal authority, and how can suspicious vehicles be excluded from traffic?

RQ5 How should SDVs respond to security incidents while in motion?

Is it feasible to predict the worst-case safety consequences if a security-aware SDV equipped with reactive and proactive defense techniques reports a security alarm? Is it obligatory for an SDV to always come to a safe halt in response to a security incident? How can a potential assessment scheme account for the propagating effects and identify attack paths, such as an attack originating from an initially inconspicuous position but ultimately impacting safety-critical components? Chapter 8 tries to provide answers to these questions.

1.5 Contributions

Our research questions resulted in security solutions that constitute our main contributions. In this dissertation, we followed a holistic engineering approach for the secure design and operation of an SDV. The term *holistic* emphasizes our efforts to cover the entire vehicle lifecycle with security awareness, starting in the early conception phase and extending to vehicle operation in real-world traffic. As outlined earlier, we anticipate that future road vehicles, particularly those capable of autonomous driving, will incorporate novel software and E/E architectures while well-established legacy technology prevails at specific points. Therefore, this work not only focuses on securing emerging technologies but also addresses the security needs of existing legacy systems. Our holistic approach considers security a process rather than merely a set of features added to an existing platform. Therefore, besides presenting specific defense techniques, our work defines a cost-efficient and scalable maintenance process for a modern automotive software architecture. Security goes beyond encrypting traffic and detecting intruders; it must remain scalable, maintainable, and certifiable within the software architecture, or it is unlikely to be deployed in a commercial product.

Our scientific contributions begin in the conceptual phase with a systematic risk analysis of a software-driven road vehicle, which we develop as part of a large consortium of researchers, as described in Section 1.6. We present a consensus-based implementation of the generic ISA-62443 cybersecurity standards for the UNICARagil vehicle platform. In that context, we identify risk evaluation criteria, suggest a scoring scheme to assess automotive risks, and propose a hierarchical thread model for a collaborative threat identification process. Based on the identified security requirements, we then present various techniques for protecting signal-based CAN and FlexRay networks. In particular, we focus on efficiently managing cryptographic keys in legacy networks and defense techniques against traffic manipulations. Regarding emerging Ethernet-based service-oriented middlewares, we define a cost-efficient, maintainable security process for the recently presented Automotive Service-Oriented Architecture (ASOA). This process is designed to be transparent from functional developments and enable software module relocations within the SDV. Our security solution secures service-oriented communication while maintaining the loose coupling of software modules and separating system knowledge and functions. While these works assume an attacker controlling the network, we present a proactive scheme to detect the corruption of ECUs. This scheme determines the software integrity of an ECU in a secure manner, allowing the SDV to prove its trustworthy state to a third party. That way, authorities or service providers can validate a vehicle's software state and grant further actions, such as participation in automated traffic.

Lastly, we focus on a vehicle's operating phase, presenting a scheme that assesses the severity of security incidents. We assume a security-aware SDV that incorporates all countermeasures derived during the initial risk analysis. However, we argue that a protected vehicle still needs the means to react to security alarms, such as when a defense technique detects suspicious behavior. Our scheme assesses the worst-case severity of reported security incidents, enabling the vehicle to perform appropriate safety actions.

All contributions are presented in six primary scientific papers [1, 2, 3, 4, 6, 7]. Each chapter of this thesis is based on at least one publication mentioned at the outset of the respective chapter. While all contributions were thoroughly evaluated in realistic environments, some were additionally integrated into the aforementioned prototype vehicles. These vehicles resulted from the UNICARagil project and were unveiled to the public in May 2023.

1.6 UNICARagil

In 2018, a consortium of eight German universities[1] and selected industry partners received funding from the Federal Ministry of Education and Research for the *UNICARagil* project. The project aimed to design a platform for a fully automated and electric SDV and to construct four prototype vehicles based on that common platform. The project aimed to address the disruptive mobility trends and technological challenges arising from the ongoing transformation from conventional automobiles to automated SDVs.

While the automotive industry naturally reacts slowly and cautiously when it comes to adopting utterly new vehicle concepts, academia has different possibilities and incentives, as it does not need to produce and sell a series of vehicles. The objective of the UNICARagil project was to consolidate highly specialized knowledge and expertise from leading German universities into one vehicle platform, fostering collaboration instead of competition. As pointed out earlier, Germany's economy is highly dependent on the automotive industry, which is undergoing fundamental changes in an increasingly competitive and international environment. UNICARagil attempts to bundle state-of-the-art knowledge in a prototype vehicle to pave the way for the industry and to showcase what an automated SDV may look like. The political goal behind UNICARagil is to strengthen Germany's industrial base.

UNICARagil utilizes a modular and scalable vehicle concept consisting of a common platform for adding customized superstructures. This modular approach significantly reduces cost, as new vehicles need not be designed from scratch. UNICARagil demonstrates the applicability of that platform with four distinct prototype vehicles: The *autoElf* is a privately owned vehicle, while the *autoTaxi* can be ordered on demand for driving services. In contrast, the *autoShuttle* is designed to transport human beings, particularly in rural areas with limited public infrastructure. Finally, the *autoCargo* is intended for the automated delivery of parcels using a robotic arm. All vehicles operate at SAE Level 4 automation and are displayed in Figure 1.1.

Figure 1.1: The four UNICARagil vehicles: autoShuttle, autoTaxi, autoElf, autoCargo

The vehicles incorporate novel concepts covering nearly the entire spectrum of automotive engineering areas. This includes an efficient battery management concept for a stable and economical power supply, individual 90-degree wheel actuation, allowing for sideways maneuvers, and a central control unit at the heart of each vehicle. Among others, this control unit executes the vehicle's controller for converting trajectories into low-level driving commands and contains the self-perception unit, a crucial safety system component. This unit continuously monitors and

[1] RWTH Aachen, TU Darmstadt, University of Passau, TU Munich, TU Braunschweig, University of Stuttgart, University of Ulm, Karlsruhe Institute of Technology

assesses the vehicle's capabilities, initiating measures to ensure passenger safety when necessary. Another fundamental safety mechanism is the provision of an emergency trajectory implemented reflexively in case of failures, which typically results in a safe halt.

The vehicle platform is organized as a zonal E/E architecture, employing fewer but comparatively powerful ECUs. The control units use the ASOA to process and exchange data in real-time. A sophisticated sensor system, including cameras, lidar, and radar components, ensures a 360-degree perception of the surroundings, resulting in a detailed environmental model. This model is enriched with real-time information from the road infrastructure, enabling route planning algorithms to optimize traffic depending on the current context. Additionally, a remote control unit allows human intervention if automation cannot resolve an unexpected situation.

The complexity of the proposed vehicle platform led to numerous scientific publications in various research areas, such as automation, modularization, validation, safety, and security. As a partner of the UNICARagil consortium, we contributed a holistic security concept to protect SDVs from cybersecurity attacks. Therefore, the research questions in this dissertation align with the UNICARagil project's objectives. Our security considerations influenced the design and development of the UNICARagil vehicle platform from its early phases. In particular, the security process and defense mechanisms presented in Chapter 6 were fully integrated into all four vehicles. Yet, our solutions are not specific for the UNICARagil vehicles but applicable to any SDV.

On May 11, 2023, the UNICARagil vehicles were unveiled to the public, who had the opportunity to take automated test drives at the Aldenhoven Testing Center.

1.7 Thesis Outline

In Chapter 2, we provide essential background information to understand the subsequent chapters comprehensively. After that, we present an overview of related work for each research question in Chapter 3.

Chapter 4 examines the applicability of the ISA-62443 standards to automated, software-driven road vehicles, comparing them with the novel ISO/SAE 21434 standard. We will systematically identify and assess cybersecurity risks through a semi-automated process, leading to the derivation of security requirements for the UNICARagil vehicle platform. To do so, we introduce the UNICARagil hardware and software architecture as our reference platform. Throughout this work, we will present technical solutions for selected requirements and integrate them into the UNICARagil vehicles to reduce the overall security risk to a tolerable level.

In Chapter 5, we will address the security requirements of classical automotive signal-based communication protocols. Specifically, we will present a scheme for efficiently deriving cryptographic group keys for the CAN protocol using implicit certificates. Subsequently, we will investigate highly critical FlexRay-based communication and present methods to protect it from manipulations.

Chapter 6 will shift the focus toward securing emerging automotive middleware stacks using service-oriented communication. We will define a security process for the novel ASOA, enabling cost-efficient specification and maintenance of secure system-agnostic services on both embedded and high-performance platforms. We will demonstrate that our solution can quickly adapt to software updates and relocations within the vehicle while remaining transparent from functional software developers. This process was integrated into all four vehicles, making them capable

of repelling network attacks, but it was also adopted and tested by the UNICARagil consortium.

In Chapter 7, we will assume the presence of attackers residing on in-vehicle control units, thereby controlling the software without necessarily corrupting the network. This assumption is based on our security analysis from Chapter 4, which revealed that the ability to receive remote updates and allow for software customizations makes software attacks likely. We will present a proactive scheme that allows a vehicle to prove its trustworthiness to a service provider, such as a legal authority, a vehicle fleet, or a highway operator. This approach ensures that automated traffic is not disrupted by malicious vehicles.

Chapter 8 will introduce a scheme enabling SDVs to react to security incidents during operating time. Once a vehicle has been equipped with both reactive and proactive security measures, it will need the ability to assess the occurrence of security alarms, particularly during runtime. These alarms may be false and accidental, e.g., resulting from a transmission error, or indeed indicate an actual attack. Depending on the vehicle context and its internal layout, the expected severity of the consequences of a security incident can range from negligible to fatal. Our work will analyze how an attack may propagate through the vehicle, eventually leading to a safety response.

Finally, in Chapter 9, we provide a summary and conclusion of our work, along with an outlook on further advancements and unresolved questions in the field of secure SDVs.

2. Background

This chapter provides essential background information for the comprehensive understanding of this dissertation. We specifically introduce protocols, standards, and frameworks that will be used throughout this work.

2.1 Software-Defined Vehicles

A Software-Defined Vehicle (SDV) predominantly enables its functions and features through software, unlike traditional vehicles that use dedicated hardware modules to accomplish the same. This software-centric approach enables manufacturers to replace, update, or upgrade the vehicle's software after production. SDVs, while not necessarily fully self-driving, typically incorporate a wide range of automation features. An SDV may sometimes integrate legacy technology in specific areas, though its core architecture is usually characterized by a modular, service-oriented design. A holistic security strategy is essential to safeguard SDVs against cyberattacks. This strategy must encompass the entire system rather than focusing solely on protecting individual components. The UNICARagil vehicle, as introduced in Section 1.6, serves as an illustrative example of an SDV.

2.2 Controler Area Network

The Controler Area Network (CAN) is a multi-master broadcasting protocol used for prioritized and event-triggered data transmission in automotive systems. Originally developed by Bosch in 1983, it has been standardized as ISO 11898 since 1994 [24]. Its cost-effectiveness, robustness, efficiency, and flexibility have been key factors contributing to its success. Today, CAN is primarily employed in road vehicles but can also be found in drones, radar systems, and even submarines.

CAN is a message-based protocol, meaning frames do not have dedicated recipients. Instead, they are identified by arbitration identifiers that define their message type. Consequently, when receiving data, CAN controllers filter messages based on their types. The arbitration identifier also determines the frame priority on the CAN bus. Data are encoded in dominant and recessive bits, with dominant signals automatically taking precedence over recessive ones. Frames with lower arbitration identifiers have a higher priority.

Data is transmitted over a twisted pair of wires, where a dominant bus level corresponds to a logical 0, and a recessive level is interpreted as binary 1. Since dominant signals are inherently prioritized on the bus, typically, safety-critical commands are sent with higher priority.

In 2012, an enhanced version of the classical CAN protocol, known as CAN with Flexible Data rate (CAN-FD), was introduced to address the growing demand for higher bandwidth in auto-

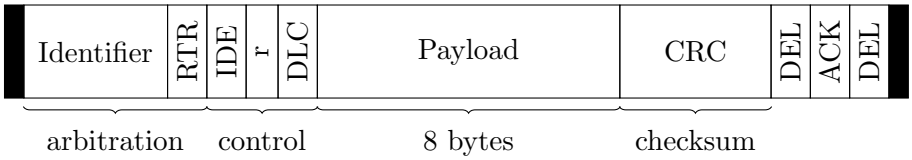

Figure 2.1: Classical CAN frame. The arbitration identifier defines the CAN message type and likewise determines the frame priority. RTR indicates whether the sender requests rata. DLC encodes the message length and DEL are delimiter bits.

motive networks. CAN-FD supports a theoretical bandwidth of up to 8 Mbit/s, whereas classical CAN allows a maximum of 1 Mbit/s. Since CAN was initially designed for the periodic transmission of relatively small data packets, each classical CAN frame has a limited payload of 8 bytes. In contrast, CAN-FD frames can carry up to 64 bytes. Figure 2.1 illustrates the simplified structure of a classical CAN frame.

2.3 FlexRay

FlexRay is a time-triggered in-vehicle communication protocol suitable for highly safety-critical applications demanding robust and deterministic real-time behavior. It is commonly deployed in x-by-wire systems, Active Cruise Control (ACC), and the Anti-lock Braking System (ABS). Since 2013, FlexRay has been defined by the ISO standard 17458 [25].

In contrast to the well-known CAN protocol, a FlexRay channel provides higher data throughput (up to 10 Mbit/s instead of 1 Mbit/s) and a higher degree of reliability (TDMA-based messaging). Additionally, a second physical link allows for fault-tolerant data transmission.

FlexRay communication is divided into periodic cycles of fixed duration. A cycle adheres to a predefined communication schedule containing all network parameters known to every node. The general underlying time unit is called macrotick, typically lasting between 1μs and 6μs. The overall duration of a complete cycle usually ranges between 8 and 16,000 macroticks. As illustrated in Figure 2.2, each cycle consists of at least two segments: the static segment and the Network Idle Time (NIT). The static segment enables communication in a TDMA fashion, ensuring reliable data transmission and fixed network latencies. It is divided into a fixed number of time slots (*gdNumberOfStaticSlots*), known as static slots. A static slot, in turn, consists of a predefined number of macroticks (*gdStaticSlot*). The total duration of the static segment can be expressed as the product *gdNumberOfStaticSlots* × *gdStaticSlot* × *gdMacrotick*. All static slots have the same length and cannot be omitted during a cycle. A null frame is sent if a node has no data to transmit in an assigned static slot. This ensures continuous traffic inside the static segment. To keep track of the current slot, all nodes maintain a counter *vSlotCounter*,

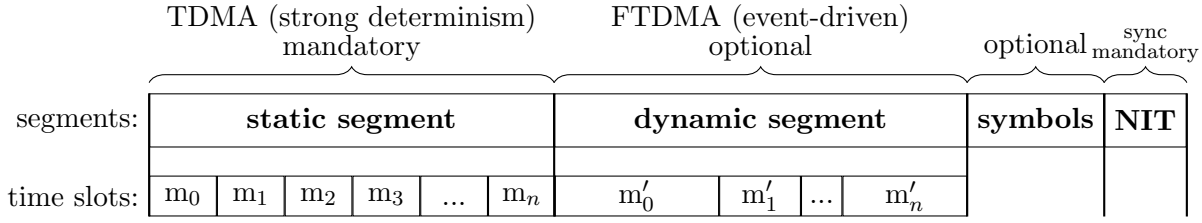

Figure 2.2: Structure of a FlexRay communication cycle

incremented by one at the end of every slot. The NIT allows node synchronization, which is crucial for enforcing the communication schedule.

A communication cycle can optionally contain a dynamic segment and a symbol window. The dynamic segment enables sporadic communication and is likewise composed of time slots called dynamic slots. A dynamic slot consists of minislots lasting between two and 63 macroticks (*gdMinislot*). Similar to static slots, dynamic slots are assigned to network nodes. However, they are not of fixed length but adjust dynamically in size. The duration of a dynamic slot extends until the end of the minislot, at which point a given node terminates transmission. Thus, once a node has permission to transmit on a dynamic slot, it suppresses the transmission of other nodes, enabling data prioritization similar to CAN, where high-priority data supersedes low-priority data. The communication duration is limited only by the predefined length of the dynamic segment (*gdNumberOfMinislots*). In case no node transmits data, the length of each dynamic slot defaults to one minislot. The symbol window indicates specific activities inside a FlexRay network, such as the start of communication, with three symbols: the Collision Avoidance Symbol, the Media Test Symbol, and a Wakeup Symbol.

Figure 2.3 shows the structure of a FlexRay frame, consisting of a header, a payload, and a trailer. While the trailer contains a CRC over the entire frame, the header integrity is additionally ensured in a dedicated header CRC field.

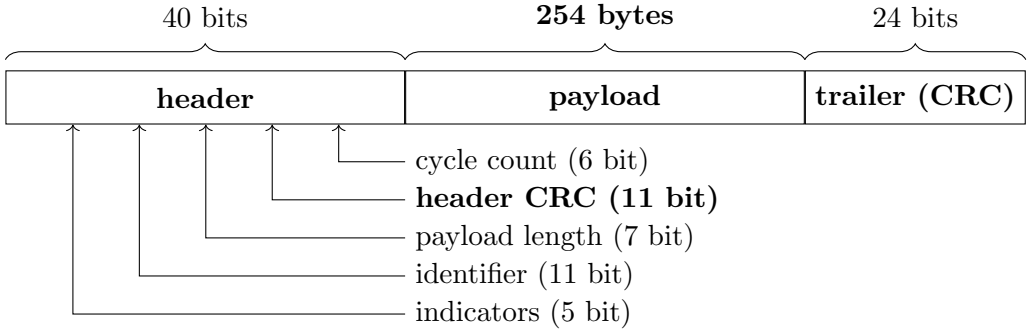

Figure 2.3: Simplified structure of a FlexRay frame.

The FlexRay protocol provides a redundant channel layout to address highly safety-critical applications and to increase fault tolerance. A second, electrically decoupled channel can either transmit data redundantly or double the bandwidth to 20 Mbit/s. The dynamic transmission may differ, while both channels' static segments look identical. This is why two slot counters are maintained on each node for the dynamic segment.

2.4 SOME/IP

The Scalable Service-Oriented Middleware over IP (SOME/IP) [26] is designed for efficient data transmission among heterogeneous control units in Ethernet-based automotive networks.

As the name implies, it introduces the design principle of service orientation to enhance flexibility and interoperability in road vehicles. A service typically represents a software component offering a specific function. Services across the system interact by offering and consuming data. In contrast to classical signal-based protocols, where data is sent once available, SOME/IP transmits data only when needed, resulting in more efficient bus usage. Nevertheless, it is designed to coexist with signal-based protocols and allows the transport of CAN and FlexRay frames.

The middleware comprises three fundamental building blocks: SOME/IP, SOME/IP-SD, and SOME/IP-TP. The first one, SOME/IP, defines message formats, contains rules for communication between services, and prescribes data serialization. SOME/IP-SD adds service discovery capabilities to SOME/IP networks. During this process, services can dynamically discover and register each other. A service can be identified by a service ID, while each service instance has a unique instance ID. That means the same service can run multiple times in a network, allowing different servers to offer the same function. The `OfferService(service ID, instance ID)` sends a multicast message indicating that a server offers a specific service to the network. While the service ID needs to be precise, the instance ID may be filled with the placeholder 0xFFFF if no specific instance is necessary. Respectively, the `StopOffer(Service ID, Instance ID)` tells the network that the corresponding service is not available anymore. If a client does not receive a required service, the `FindService(Service ID, Instance ID)` enables them to search for a service providing specific data. Hence, services require system knowledge since the service ID and the instance ID are necessary to create such requests. The SOME/IP middleware allows servers to define a priority and a weight for its service such that clients can decide for a specific instance in case different servers offer the same service. Additionally, servers allow load-balancing options to optimize the bus usage.

SOME/IP services can implement a publish-subscribe pattern for asynchronous data exchange. The `SubscribeEventgroup(service ID, instance ID)` allows a service to subscribe to another service, automatically receiving notifications about specific events. Similarly, a subscription can be stopped with the `StopSubscribeEventgroup` command. Besides the publish-subscribe approach, SOME/IP offers Remote Procedure Calls, enabling a service to execute a remote function and request the output.

The last building block, SOME/IP-TP, adds support for segmented or fragmented messages. This feature is used to segment large packets which do not fit into a single UDP packet. In case latency does not play a crucial role, it is possible to use SOME/IP over TCP.

The SOME/IP specification does not incorporate security features, leaving communication security to the system operator. Therefore, SOME/IP networks are usually secured on the link layer using MACSec or the AUTOSAR SecOC [19].

2.5 Data Distribution Service

The Data Distribution Service (DDS) [27] is a middleware standard for reliable, scalable, and efficient real-time communication. Maintained by the Object Management Group (OMG), DDS has found significant applications in embedded systems, particularly in domains such as automotive, military, aerospace, and robotics, where real-time data exchange is crucial. Although, initially, DDS has not been developed exclusively for the automotive domain, the AUTOSAR, a worldwide association between automotive industry partners, integrated DDS into their adaptive platform as a connectivity standard in 2018. DDS utilizes a topic-based publish-subscribe pattern for data transport between writers and readers. Notably, DDS distinguishes itself from other publish-subscribe protocols like MQTT [28] by operating in a fully decentralized manner, eliminating the need for a central broker. Instead, DDS participants discover each other by periodically sending out beacons and keeping track of the network topology. DDS defines key entities such as topics, domains, publishers, and subscribers. A topic describes a specific data type to which Quality of Service (QoS) parameters can be appended. That way, enforcing real-time, redundant, bandwidth, resource, or cached policies is possible. These capabilities enable

DDS to tailor its behavior to the application's specific needs. DDS participants operate within domains which delineate the communication context for DDS entities. Participants within the same domain can communicate with each other through the exchange of data using publishers and subscribers. Publishers are responsible for managing DataWriter objects, while subscribers utilize DataReaders to receive and process data. To ensure interoperability among different DDS implementations, the Real-Time Publish-Subscribe (RTPS) wire protocol has been developed and standardized [29]. This underlying protocol facilitates the discovery, communication, and reliable data delivery between DDS participants, regardless of their specific implementations.

2.5.1 DDS Security Specification

In 2014, the OMG added a security specification [30] to prevent tampering with DDS traffic. This specification comprises five Service Plugin Interfaces, which offer out-of-the-box security capabilities encompassing authentication, access control, cryptography, logging, and data tagging. Secure DDS entities employ symmetric cryptography to ensure the confidentiality and integrity of their data exchanges. However, the decentralized nature of DDS, coupled with the absence of a central authority responsible for security, introduces distinct infrastructure requirements and influences the behavior of DDS entities. To establish secure DDS communication, entities must acquire and manage relevant key material from authorized remote entities. The authentication of DDS traffic necessitates the possession of at least two X.509 certificates by a domain participant: The first certificate defines the Identity Certificate Authority (CA), while the second one binds the participant's GUID, the Global Unique IDentifier to its public key. Additionally, another certificate from a Permissions CA is required for implementing access control. This CA signs governance and permission documents, which need to be maintained by each participant. During the discovery phase, a three-way handshake between pairs of DDS participants ensures the secure transmission of key material through the *Secure Key Exchange Channel*. The sender and the receiver must fully match, encompassing identical topics, security attributes, and compatible QoS settings, as a prerequisite of this handshake. Throughout this process, the sender and receiver mutually authenticate using the earlier-mentioned certificates, exchange signed permission files, and share security attributes. To verify an incoming message, the receiver utilizes the sender ID and key ID to identify the corresponding key from its locally stored key material. This key, in combination with a session ID, allows the receiver to derive a session key for message verification. As the session key is only valid for one communication channel, the number of keys rises proportionally with the number of topic subscriptions.

2.6 Physical Unclonable Functions

Electronic devices exhibit unique physical properties that can be leveraged to derive a digital fingerprint, which can be transformed into a cryptographic key. These properties depend on random and unpredictable factors, such as minor variations in transistor doping [31].

For a particular input (challenge), a Physical Unclonable Function (PUF) should consistently generate the same output (response) on the same device. Consequently, a challenge sent multiple times to the same device must lead to the same response, while other devices create different responses. A fuzzy extractor scheme can mitigate noise in the response, stabilizing it and converting it into a usable key. To facilitate key sharing, a one-time enrollment process is required. During this process, party A gains access to the challenge-response pairs of party B's PUF and

selects a fuzzy extractor scheme. Conducting this enrollment process in a secure environment is crucial to prevent the leakage of challenge-response pairs.

As explained in [32], A subsequently can select a key and encode it by utilizing a PUF response through the chosen fuzzy extractor scheme, resulting in what is referred to as helper data. Typically, this helper data stabilizes the PUF response by eliminating erroneous bit flips. Since the helper data does not contain confidential information, it is not required to be kept secret, except for the challenge-response pairs. When A wishes to share this key with B, it transmits the helper data and the challenge c to B, where c corresponds to the PUF response. B can then reconstruct the key chosen by A by decoding the helper data using the PUF response associated with c through the fuzzy extractor scheme.

Consequently, a PUF can retrieve and share unique keys from hardware. Intrinsic PUFs are based on the inherent properties of system components, such as memories, and do not necessitate additional hardware. Memory-based PUFs exploit the distinctive behavior of memory cells. For example, SRAM-based PUFs primarily build upon the uniqueness of SRAM cell values after a device reboot, while certain DRAM PUFs are based on the decay of the values in DRAM cells that are not being refreshed. As a result, commonly used memory components can be employed to generate device-specific keys.

3. Related Work

This chapter outlines the scientific context in which this dissertation takes place. While Section 3.1 discusses cyberattacks on road vehicles, each subsequent section presents relevant related work for our five research questions. We highlight each work's main contributions and differences, which we attempt to address in our research.

3.1 Cyberattacks on Road Vehicles

The following attacks are just a few that demonstrate the vulnerability of vehicles to cyberattacks. What they all have in common is exploiting multiple vulnerabilities to get into safety-critical areas. Once this is possible, an attacker can control the engine, block critical communication, and provide passengers with false information. Some of the vulnerabilities were known and resulted from outdated software. As long as vehicles had no external interfaces and did not communicate with remote parties, the "security by obscurity" approach was effective.

Today, it is essential to systematically identify a vehicle's attack surface, detect attack paths, implement effective protective measures, and be able to handle security incidents at runtime.

Miller and Valasek [18] successfully infiltrated a Jeep Cherokee in 2014, taking complete control of the vehicle while in motion. The attack consisted of a chain of malicious events, eventually allowing them to inject driving commands to a safety-critical CAN bus. More precisely, they first broke into a poorly secured infotainment system, then manipulated the firmware of a gateway, and eventually injected driving commands into the bus.

The entry point to the vehicle was a vulnerability inside the infotainment system. This system provides a WiFi network whose password was set based on the manufacture day and, thus, was predictable. After gaining access to the infotainment system, the authors could control the music player and even get access to the vehicle's GPS position. However, ironically, the infotainment system was not directly connected to the CAN bus transporting safety-critical driving commands for security reasons. Instead of directly communicating on that bus, it could communicate with a gateway, a V850 microprocessor, allowing it to listen to the CAN bus but not to send commands. The authors succeeded in manipulating this gateway's firmware by simply upgrading it. No authorization checks were necessary for that step.

By taking control of that chip, the authors could finally send safety-critical commands through the CAN, enabling them to control the entire car, including the steering wheel, the engine, and the braking system.

This attack impressively demonstrated how a combination of vulnerabilities on different, sometimes non-critical, vehicle components can be exploited.

Cai et al. [33] uncovered a series of vulnerabilities in BMW vehicles, enabling attackers to execute arbitrary code on in-vehicle components and send fake diagnosis messages to ECUs. The authors illustrated the risk of attack chains that exploit multiple vulnerabilities. They present attacks requiring physical contact with the vehicle and those that can be remotely triggered.

The identified vulnerabilities were located in the infotainment system, the telematic communication box, and the central gateway. These components communicate through Ethernet, CAN, and USB buses. The central gateway is linked to additional bus systems, including CAN, MOST, and FlexRay. At the same time, the telematic communication box has Internet access and can receive updates over the air.

The authors combined logic flaws, missing security checks, and software bugs to execute attack chains. For instance, by installing a fake GSM base station, they could send an SMS message to the telematic communication box, triggering a provisioning update. Despite the encapsulation of a digital signature, reverse engineering exposed a missing boundary check on a buffer containing the signature. Consequently, conducting a return-oriented programming attack became possible, ultimately allowing remote execution of arbitrary code on the telematic unit.

Once this was possible, the attack could propagate through the vehicle, allowing the resetting of ECUs and control over the driver seat. This work demonstrates how vulnerabilities can be combined, emphasizing that adversaries can infiltrate critical vehicle parts through seemingly uncritical components. All identified vulnerabilities were reported to BMW and fixed shortly afterward.

Nie et al. [34] conducted a remote attack on the Tesla Model S by exploiting a complex chain of vulnerabilities. Initially, they accessed the vehicle through the cellular network and subsequently compromised multiple in-vehicle components, including the instrument cluster, the central information display, and a gateway.

An outdated and insecure web browser enabled the opening of a remote shell for arbitrary code execution. An additional kernel vulnerability led to a privilege escalation necessary for the successful execution of any code. From there, the authors demonstrated that the instrument cluster accepted incoming SSH connections without asking for a password.

Furthermore, the authors took control of the gateway by obtaining another shell. Reverse engineering the gateway's firmware exposed a security token that was left unprotected in the code. This token was utilized to open the shell, enabling the transmission of arbitrary CAN message on the in-vehicle bus system. This allowed the vehicle's lights to be turned on or ESP messages to be blocked, posing a dangerous intervention for the passengers.

Similar to previously mentioned attacks, a combination of vulnerabilities affecting the browser, the kernel, and ECU firmware resulted in an attack that seriously jeopardized passenger safety.

3.2 Security Requirement Analysis

Security has not been a priority in developing vehicles or their components until recently. The first cybersecurity standard for the automotive sector, ISO/SAE 21434 [35], was published only in 2021 in response to the UNECE regulation R155, which mandates cybersecurity certification. Only a few works address a systematic requirements analysis in the automotive sector, some of which are presented below.

Schmittner et Macher [36] provide an overview of safety and security engineering standards and discuss their applicability in the automotive domain. The IEC 61508, first released in 1998, introduces a safety life cycle and a probabilistic failure approach for safety-related systems. Based on that, ISO 26262 is a functional safety standard tailored for the automotive sector. Much like the IEC 61508, it evaluates risks of hazardous situations, helping to derive safety measures to avoid failures. The standard outlines a safety lifecycle and categorizes safety requirements using Automotive Safety Integrity Level, with its second version incorporating recommendations for a safety and security co-engineering approach.

Regarding security, the SAE J3061, first published in 2016, is a collection of guidelines governing the secure development of automotive systems. It suggests high-level recommendations, including a lifecycle process and information on existing methodologies and tools. Notably, SAE J3061 has significantly impacted the development of the ISO/SAE 21434, the first complete cybersecurity standard for road vehicles. Like ISO 26262, it employs integrity levels, so-called Automotive Cybersecurity Integrity Levels, to quantify security requirements. Since the publication of ISO/SAE 21434, the SAE J3061 has been withdrawn, although there are expectations for its extension with additional technical documents addressing aspects beyond the scope of the ISO/SAE 21434. While the latter standard is primarily tailored for road vehicles, the ISA-62443 and the ISO 27000 present a broader security requirement analysis process encompassing cyber-physical systems.

Beyond standardization, the authors underscore the increasing need to integrate cybersecurity into regulatory frameworks. In this context, the UNECE, which establishes rules for type approval across 62 countries, mandates incorporating a cybersecurity management system and a secure process for software updates.

In summary, the authors illustrate the growing security awareness in the automotive sector and highlight the endeavors towards combining security and safety aspects in an automotive engineering process.

Steger et al. [37] presented the DEWI metric for a systematic security analysis of a cyberphysical (automotive) system. This metric is primarily designed to support the SAE J3061 guidelines to identify security requirements for an automotive system during the conceptual phase. It can be seamlessly integrated into the SAE J3061 concept phase. Before applying the DEWI metric, the overall system needs to be subdivided into subsystems, each consisting of components. Every component is associated with a set of parameters, such as cryptographic primitives. The authors suggest a non-linear scale to assess the system parameters, arguing it aligns better with human interpretation of criticality.

When applying the metric as part of the SAE J3061, a Threat Analysis and Risk Assessment (TARA) is typically one of the first steps to be conducted, resulting in a list of potential threats associated with a distinct security level. The authors propose a scheme to map this security level to a DEWI security goal, thereby achieving a secure system configuration that is eventually transformed into precise requirements.

While the necessity for meaningful and reproducible assessment tools during a security requirement analysis becomes evident, the proposed metric is only explained at a high level, making it challenging to adopt it in an actual risk assessment.

Schmittner et al. [38] apply the SAE J3061 to an exemplary in-vehicle gateway, conducting a threat analysis and defining high-level security requirements to mitigate potential threats. The

SAE J3061 adopted concepts from the well-established ISO 26262 safety standards, such as the characteristic V-model as a security lifecycle.

The example automotive gateway, for which the authors envision a security-aware development stage, connects to various physical buses such as Ethernet, USB, CAN, GSM, and WLAN, supporting functionalities like remote control, maintenance tasks, and over-the-air updates.

Throughout a TARA, a fundamental component of the SAE J3061, the authors identify *software*, *functionality*, and *data* as assets, pointing out that these assets must align with business goals and regulatory requirements. In a brainstorming session involving a team of experts, they identify potential attack scenarios and associated threats. Each threat is linked with possible consequences, with a focus on safety and security aspects. Furthermore, they calculate an attack probability by adding up attack potentials, which include factors such as time, expertise, knowledge, window of opportunity, and equipment. This assessment approach is adopted from the HEAVENS [39] project. Ultimately, they identify high-level security goals from which precise functional requirements can be derived.

The authors conclude that identification and authentication control, secure communication, system integrity, and the management of cryptographic keys are essential requirements for securely constructing the automotive gateway in accordance with SAE J3061 guidelines.

3.3 Securing the CAN communication

The Controler Area Network (CAN) is one of the most well-known signal-oriented communication protocols. Its widespread use in vehicles has repeatedly exposed its vulnerability to attacks. As a result, numerous publications have emerged, primarily proposing mechanisms for manipulation security. Some of these works present cryptographic solutions to ensure protection against traffic manipulations. Others focus on key delivery, and some aim to identify traffic anomalies to detect adversaries. While Section 2.2 provides technical insights into the CAN protocol, we explain the scientific context of research question RQ2 in the following.

Jain and Guajardo [40] leverage the physical properties of the CAN bus to compute shared keys between ECUs. They intentionally allow data collisions to occur on the CAN bus and derive a secret key based on the arbitration results. In the initial run, two parties simultaneously transmit data on the CAN bus, leading to a collision. In the second run, the data is inverted and again transmitted simultaneously. In both cases, the output on the CAN bus is recorded. The bit positions where both recordings are 0 are concatenated to produce the secret key, which an adversary cannot reproduce. This scheme is also extended to the multi-party scenario. The authors present a thorough analysis but need to provide a practical implementation. Key authenticity is ensured by assuming the existence of a pre-shared key among all ECUs and the gateway.

vatiCAN [41] is a backward compatible authentication mechanism for the CAN bus. Messages are authenticated by computing a fresh Hash-based Message Authentication Code (HMAC) transmitted in a CAN frame. Since a single CAN frame can transport a maximum of 8 bytes, the HMAC is transmitted in a dedicated frame. Consequently, only every second frame can be used for payload data transmission, halving the overall bandwidth. Therefore, the authors recommend authenticating only safety-critical commands to compensate for the bandwidth loss.

Regarding key distribution, vatiCAN assumes that all cryptographic keys are distributed during vehicle assembly. However, this approach renders the vehicle inflexible regarding key updates, which might be necessary in the event of corruption or hardware replacements. The authors explicitly acknowledge the necessity of key provisioning but still rely on manual key distribution due to the expected overhead.

LeiA [42] is a lightweight authentication protocol designed for the CAN bus. Message Authentication Codes (MACs) are utilized to ensure data authenticity and, similar to vatiCAN, they are transmitted in an additional CAN frame. Each time the vehicle starts, a session key is derived from a long-term symmetric key permanently embedded in the ECUs during the production phase. This session key is supposed to facilitate a more adaptable key management. However, if the long-term key becomes compromised, it jeopardizes the security of the entire vehicle system. Technically, the ECUs need secure memory to prevent attackers from compromising keys. We argue that key sharing at production time poses a challenge, given the involvement of various companies in ECU manufacturing.

Fassak et al. [43] use Elliptic Curve Cryptography (ECC) to establish a session key in CAN networks, subsequently employed for generating symmetric keys for authenticating CAN frames. They employ a recipient-based authentication scheme, meaning that an individual MAC is computed for each ECU. Given that CAN is a broadcasting protocol, this approach results in significant computational overhead for the ECUs. The resulting MACs are concatenated and then truncated to 1 byte, allowing them to fit alongside the data payload within the same CAN frame. At vehicle production time, the manufacturer is tasked not only with installing an individual ECC key pair on each ECU but also with including the public keys of all other ECUs. Since no key distribution mechanism is available, key updates are not possible.

Vasile et al. [44] evaluate four keying paradigms for automotive broadcast authentication protocols. Since most studies on in-vehicle traffic authentication predominantly rely on symmetric cryptography, it is crucial to establish a reasonable keying strategy. The authors categorize these strategies into four main types: global keys (I), pairwise keys (II), group keys (III), and periodic keys (IV). The most basic approach, type I, involves a single key shared among all ECUs. In contrast, type II establishes a unique key for each pair of ECUs. Type III employs a key for a group of ECUs, making it more scalable with respect to the number of ECUs and messages compared to II. Type IV proposes broadcasting cryptographic keys at fixed time intervals. The authors conclude that sharing symmetric keys among groups of ECUs represents the most realistic approach, as it strikes the best balance between security and bandwidth efficiency.

3.4 Securing the FlexRay communication

Unlike the CAN protocol, FlexRay is primarily used for highly critical communication, such as x-by-wire applications and engine control. FlexRay communication is structured into static and dynamic time slots, guaranteeing ECUs to communicate in a specific time interval. This design ensures deterministic and predictable communication. FlexRay is vulnerable to cyberattacks since there are no inherent security mechanisms to protect against manipulations. Section 2.3 provides technical insights into how FlexRay operates, while the following publications describe various efforts to enhance the security of FlexRay communication. These resources further contextualize research question RQ2 within the scientific literature.

Murvay et al. [45] demonstrated in their work that FlexRay is susceptible to both message spoofing and Denial-of-Service (DoS) attacks. They distinguish between two types of DoS attacks: full and targeted. Full DoS attacks aim to disrupt communication entirely, while targeted DoS attacks focus on suppressing specific message types, such as all messages within a given time slot. Once a malicious device gains access to the bus, it can initiate a DoS by continuously transmitting a dominant signal, preventing legitimate messages from being received and rendering communication impossible.

The authors note that message spoofing goes undetected only in the dynamic segment of a communication cycle, as a malicious node cannot occupy a foreign static slot without causing a collision, which would be detectable. Since data transmission in the dynamic segment is optional, message spoofing attacks may occur there.

To illustrate the feasibility of these attacks, the authors provide experimental evaluations conducted on hardware. They propose an active star topology as a mitigation technique against DoS attacks, as it allows the host node to disconnect network segments, effectively excluding malicious nodes. They suggest using cryptographic mechanisms regarding message spoofing but have yet to present a specific implementation.

Mousa et al. [46] adapted the LCAP framework [47] to extend FlexRay with authentication. Initially designed for authenticating CAN traffic, LCAP proved adaptable to FlexRay, benefiting from the larger payload size of FlexRay frames that negates the need for authentication tag truncation. In addition to frame authentication, LCAP also offers traffic encryption capabilities. However, one drawback of LCAP is the high number of exchanged messages required in advance to establish secure communication. Nonetheless, LCAP's advantage lies in its lack of need for hardware modifications, making it an appealing choice for a FlexRay security framework. Mousa et al. demonstrated that LCAP's implementation minimally impacts the overall system performance. Nevertheless, they emphasize the need for an effective key distribution technique. The current protocol version relies solely on a pre-shared key, which can complicate maintenance.

Vasile et al. [44] conducted a comparison of various key distribution schemes for ECUs. They categorized these schemes into four keying paradigms: (1) a single key for all ECUs, (2) a unique key for each pair of ECUs, (3) a shared key for groups of ECUs, (4) time-delayed keying.

These four options serve to balance the trade-off between security and overhead. Using a single key simplifies key maintenance and exchange. However, it poses a vulnerability in that compromising a single ECU can jeopardize the entire security of the vehicle system. In contrast, pairwise keying demands a key for each pair of ECUs, potentially resulting in a substantial overhead during key exchanges, depending on the network's size. Furthermore, achieving real-time behavior becomes more challenging if keys are delayed in delivery. This study served as an inspiration for our work in Section 5.4 on different keying techniques for FlexRay-based networks, particularly concerning slot-based communication.

3.5 Securing Service-Oriented Communication

Service-oriented approaches enable the scalable, modular, and adaptable exchange of large volumes of data, distinguishing them from signal-based protocols. The term "service orientation" does not primarily denote a specific communication method but rather serves as an umbrella term for an architectural type. As its name implies, such architecture comprises services, typi-

cally representing functional software components. These services can be dynamically loaded and managed, requiring minimal effort for subsequent system adjustments. Communication among these services often occurs through a publish-subscribe mechanism, ensuring that the services remain loosely coupled and do not need to manage each other directly. Service-oriented approaches are increasingly adopted in the automotive sector due to their benefits in scalability and modularity.

Notably, SOME/IP [26], a service-oriented middleware, has become an integral part of the standardized AUTOSAR Adaptive Architecture. However, not all service-oriented approaches inherently offer secure communication, leaving room for potential manipulation by attackers. Below, we present some efforts to enhance communication security in service-oriented architectures.

Iorio et al. [48] enhanced the SOME/IP framework with a security model to deter manipulation attacks while allowing security policy enforcement. For that purpose, they introduced three security levels that differ in criticality. While the first does not provide security protection, the second level demands authentication and integrity. The third level adds confidentiality to prevent eavesdropping. A network operator needs to associate software services with one of those levels. Eventually, only services of the same criticality are allowed to communicate with each other. The security policy mentioned above specifies what resources a given service can query from other services. It consists of rules represented as digital certificates, whose Subject Alternative Name field contains the actual rule in a URI format. Policy rules can be added, updated, or removed by adjusting the certificates accordingly. The security framework realizes a fully decentralized approach, i.e., a central component neither maintains keys nor the access policy. Instead, all rule certificates must be replicated and stored on every control unit.

The authors formally verify their security framework and prove the correctness of the proposed scheme. The distributed nature of the security solution for SOME/IP goes at the cost of maintainability and modularity since every ECU has to keep track of all certificates. Consequently, modifications of the access policy are complex because minor changes require an update of all control units. Moreover, the maintenance of many certificates may be complicated on embedded devices with limited (memory) resources. Therefore, whether the proposed security enhancement is applicable to an automotive system remains questionable.

Zelle et al. [49] demonstrate how an attacker can impersonate a server in a SOME/IP network with enabled link layer security. To prevent such attacks, the authors present and evaluate two security extensions in a practical setup.

The SOME/IP specification lacks security provisions, so it is usually deployed in networks with a secured link layer. Solutions like IEEE 802.1AE provide point-to-point traffic authenticity on Ethernet links. The assumed attacker has complete control over the network and the ability to compromise ECUs. The authors reveal two possible attacks by analyzing the security of the SOME/IP middleware using the Tamarin model prover [50].

One of the identified attacks is the copycat attack. Here, an adversary $\mathcal{A}dv$ waits for a benign server S to broadcast an `OfferService` o. Once this happens, $\mathcal{A}dv$ generates a deceptive service offer o' with its address in the endpoint options while retaining the service ID and the instance ID. Consequently, a client C receives both o and o' for the same service

Opting for o' puts $\mathcal{A}dv$ in a Man-In-The-Middle (MITM) position. In the de-association attack, $\mathcal{A}dv$ makes the client C believe that the legitimate service offer o is no longer available. This is

achieved by sending a `StopOffer` shortly after o has been put on the network. Consequently, client C no longer faces a choice between two offers for the same service and is more likely to accept o', thereby enabling $\mathcal{A}dv$'s MITM position.

Both attacks exploit a common vulnerability: a SOME/IP client does not verify the authentic origin of a service offer. Even with a secured link layer, the client cannot ascertain the origin of a service offer. The secured link layer only prevents $\mathcal{A}dv$ from direct communication in the network, necessitating the compromise of an ECU for network access.

Zelle et al. propose a decentral and a central solution to establish secure channels between servers and clients, preventing $\mathcal{A}dv$ from executing MITM attacks. In the decentral approach, servers sign service offers digitally. To implement this, all SOME/IP entities must possess public and private keys and certificates, allowing them to verify other's public keys. In a Diffie-Hellman exchange, servers and clients use their keypairs to compute a shared session key. This enables client C to quickly detect that o' does not originate from S. Alternatively, the central solution involves a dedicated authorization server that tracks the services a server can offer. Each service offer o is routed through this server, which verifies o's authenticity using a MAC and generates a session key securely transmitted to all clients. That way, $\mathcal{A}dv$ cannot fake o since the authorization server would detect this. The authors evaluate both security solutions in a practical setup, showing that the central approach outperforms the decentral one in terms of latency and message overhead.

While both security solutions effectively protect SOME/IP networks against copycat and de-association attacks, they only represent a first step towards a practical solution. In either case, a security maintenance process is necessary to distribute and update cryptographic material within the network, particularly in a dynamic software environment like that of an SDV. Furthermore, these solutions make the link layer protection redundant, as the session key enables authentication and encryption, providing similar security protection on another network layer.

Mayoral-Vilches et al. [51] present SROS2, a collection of tools to facilitate securing ROS systems. The authors present a methodology of modeling, authentication, authorization, generation, deployment, and monitoring. Since the DDS middleware has in-built security features, it must be appropriately set up to protect against cyberattacks effectively. However, a proper security setup of DDS-based environments is complex and demanding, especially in larger networks where ROS is typically deployed. SROS2 helps organize the security capabilities of ROS 2 systems in a usable way. At first, the authors propose an automated introspection of the ROS 2 computational graph to derive security policies. These are converted into authorization and authentication rules in an XML format. Subsequently, security policies are mapped to graph resources operating in the same security domain. SROS2 enables the generation of security artifacts for the previously generated policies. Finally, SROS2 provides monitoring tools to detect security flaws while running ROS 2 graphs.

Friesen et al. [52] conducted a comparative evaluation of the DDS Security Specification [30] and Datagram Transport Layer Security (DTLS) [53]. Rather than focusing on performance evaluations, their work primarily aimed to identify functional differences, configurability, and adaptability between the two frameworks. Both frameworks are technically suitable for securing a service-oriented environment. While the in-house DDS Security Specification offers mechanisms to protect DDS traffic from manipulation and unauthorized access, the DTLS likewise protects against tampering attacks but operates at the network layer instead of the application layer. Both solutions ensure essential security objectives such as authenticity, confidentiality,

and integrity. However, DDS goes beyond that by supporting non-repudiation, enhancing availability, and enabling access control. While the cryptographic capabilities of DTLS and DDS are comparable, the authors point out significant differences in configuration and adaptability. DDS allows for individual topic protection and fine-tuning connections through QoS parameters, but this flexibility introduces additional participant requirements. For instance, to ensure authenticated traffic, DDS requires the setup and distribution of multiple CAs, X.509 certificates, and signed permission and governance documents. In contrast, DTLS uses master keys for each secure connection. DDS participants establish session keys by exchanging digital certificates and utilizing a hash-based key derivation function. DTLS, on the other hand, uses the record protocol to distribute the earlier-mentioned master keys to the corresponding network nodes. While DDS enables the reuse of a master key for multiple session keys, DTLS lacks this capability. The authors conclude that DDS provides greater customization options compared to DTLS, albeit at the cost of a more complex setup involving a key management infrastructure (CAs) and the distribution of certificates and signed policy documents for each participant.

3.6 Ensuring Integrity of Automotive Software

For holistic vehicle protection, it is essential to expand the attacker model to the extent that an attacker can not only gain control of a vehicle through poorly protected communication but can also cause damage by manipulating control software. This is particularly possible as soon as software can be updated and customized.

The following publications use isolated execution environments in hardware, either to safely calculate and verify critical control commands on potentially manipulated ECUs at runtime or to measure the software integrity of a vehicle during bootup and confirm it to a trusted third party.

Erickson et al. [54] suggest the concept of autonomous vehicle contracts to enforce expected driving behavior in a platoon. A contract can be viewed as a set of driving parameters (e.g., speed, safety distance) that the vehicles in a given platoon must adhere to. According to the underlying threat model, vehicles within a platoon may be compromised, making it necessary to monitor driving commands continuously. Commands directed to the powertrain and brakes are monitored, and if they violate the contract, they are filtered. The monitoring process takes place within a secure execution environment known as an enclave. In this manner, trusted code can be executed on a potentially compromised system, enabling remote attestation that proves system integrity to a third party. Consequently, vehicles in a platoon can attest to each other's enclave, ensuring that everyone conforms to the driving contract.

Furthermore, the authors present mechanisms to respond to unexpected situations, such as suddenly emerging obstacles, where the current contract must be violated to maintain safety. In that case, the vehicle platoon first undergoes a recovery phase followed by a separation phase. During the recovery phase, the vehicle aims to recover from broken communication resulting from failures or jamming attacks. Once a timeout elapses, the platoon starts separating, which is necessary to establish a safe distance for the vehicles to regain autonomy. Only then can a driving contract be violated to respond to unforeseeable situations. For separation, the authors propose an emergency termination procedure, intending to separate vehicles as quickly as possible if communication becomes possible. This procedure calculates an individual deceleration for each vehicle, considering the platoon's size and velocity.

The evaluation demonstrates that the recovery and separation phases consume approximately 1500ms in the worst case. As this time corresponds to the human perception and reaction time, the authors conclude that their solution applies to real-world scenarios.

Kohnhäuser et al. [8] presented an attestation scheme designed to ensure the integrity of software running on ECUs. Upon startup, the scheme computes a hash of the deployed software, authenticates it using a secret attestation key, and transfers the result to a central and trusted master unit. This master unit verifies the measurements and prevents the vehicle's movement if a compromised ECU is detected. This is possible because the master unit controls the vehicle's power supply and needs to release it before the vehicle can drive.

As the attestation scheme executes in a potentially compromised environment, it necessitates specific hardware properties. Firstly, it must not be disrupted, as any disruption would allow an attacker to tamper with the integrity measurement. Secondly, it must be stored in a write-only area, preventing alterations. Thirdly, the underlying hardware must provide secure storage to safeguard the attestation key from potential attackers.

The authors present two slightly different versions of the scheme, one for *simple* and another for *advanced* ECUs. A simple ECU is an embedded device consisting of firmware loaded by the bootloader. In comparison, an advanced ECU is equipped with a full-stack OS and an ARM TrustZone for secure code execution.

By disabling interrupts on simple ECUs, the uninterrupted execution of the attestation scheme is guaranteed. In contrast, the scheme is executed on advanced ECUs within the privileged and isolated secure world provided by the ARM TrustZone. Both the integrity measurement and the verification occur inside the vehicle. While this approach is effective for a single vehicle, scalability for vehicle fleets is questionable, and maintenance in an updatable software environment poses challenges.

The authors implement and evaluate the attestation scheme, demonstrating that it does not cause perceptible delays for passengers.

3.7 Resilience of Automotive Systems

Analyzing the intrusion and propagation of attacks in computer networks is a well-explored field. Attackers are often characterized by specific abilities required to infiltrate a network and target other systems through communication channels. In the worst-case scenario, propagation effects can result in the corruption of an entire, potentially well-protected network by exploiting its weakest link. Such propagation effects are typically modeled with (attack) graphs, a method originating from fault trees that allows the identification of the cause of a failure. Typically, a node represents an attack or an adverse event caused by an attacker. Deductive analysis facilitates identifying causal relationships and examining how attacks can propagate through the network.

Road vehicles consist of networks of communicating control units, rendering them susceptible to attacks and their propagation. For instance, the well-known Jeep attack described in Section 3.1 could only achieve its full impact through propagation effects. Therefore, established concepts for attack detection and analysis need to be adapted to the vehicular environment.

Below, we present related work addressing attack detection, modeling, and analysis in computer networks. This literature presents the scientific context for answering research question RQ5 regarding the optimal response to a security incident.

Nikoletseas et al. [55] explored four schemes modeling attack propagation within computer networks. In their work from the early 2000s, the authors assumed a greedy intruder attempting to infiltrate as many systems as possible. They suggested modeling the success of an attack through spread and traceability factors. The spread factor describes the number of nodes captured by an attacker, while the traceability factor denotes the number of links between the node where an attack occurs and the node of intrusion. The attacker aims to maximize both factors to cause the greatest possible damage while making it challenging for the defense mechanism to protect the system. During their study, the authors assumed an attacker of limited power operating in a secured network.

The authors presented a new system intrusion and attack propagation model using two parameters besides the number of network nodes. The first parameter represents the probability that an attack fails and the security level of the attacked system. The second parameter describes the maximum number of attempts an intrusion software can make before being detected. For three attack schemes, the authors demonstrated that the spread and traceability factors are mostly linearly related during an ongoing attack. This implies that the efforts of intrusion detection algorithms correlate proportional to the number of corrupted network nodes.

Noel et al. [56] introduce a model for quantitatively analyzing a network's security to balance reducing vulnerabilities and managing costs. The goal is to provide means to determine the most effective security measures while keeping costs low. The authors argue that counting the number of vulnerabilities is insufficient to assess a network's security effectively; instead, a more realistic metric is necessary to represent the dependencies between them. The authors use attack graphs to describe the incremental network penetration and propagation likelihood. Specifically, they attribute probabilistic values to exploits, creating chains based on their relationships. An attack is a specific path within the attack graph, with edges possessing pre- and postconditions that must be satisfied. Attack paths, in turn, encompass multiple malicious operations an attacker conducts.

The authors employ Monte Carlo methods to simulate attack graphs under uncertain input values. While the attack graph is only computed once, it undergoes evaluation for each Monte Carlo sample. Ultimately, the metric quantifies a network's overall security and risk.

Besides the attack graph, the risk model comprises a cost analysis. This analysis assesses the expected costs for eliminating attack paths from the networked system, a crucial step in reducing overall cybersecurity risk. However, such actions may again incur substantial costs. For example, blocking SSH might effectively counter specific attacks, but it introduces complexities to network usage and could lead to maintenance problems. The author's model allows system operators to determine the most effective security measures while keeping costs low.

Roschke et al. [57] designed a correlation algorithm capable of detecting multiple attack scenarios in a computer network. The work is based on attack graphs that model existing vulnerabilities in the system under investigation. The starting point is an existing algorithmic approach that aims to reduce the number of false positives of intrusion detection systems by correlating and clustering alerts. However, this approach only considers the last alert of a particular type, making

it difficult to identify similar attack scenarios for forensic purposes. Consequently, they propose an improved algorithm consisting of five fundamental steps:

Following an initialization step, alerts are mapped to nodes of attack graphs. Next, the alerts are clustered, resulting in an alert dependency graph, enabling the detection of suspicious graph subsets with the correlation algorithm. In the last step, the authors employ a Floyd-Warshall algorithm to find the shortest paths.

Ultimately, the authors implement and evaluate their algorithm on a multi-core platform, revealing that the path-finding step consumes roughly 85% of the processing. The authors demonstrate the applicability of the proposed algorithm for forensic purposes; however, whether it can be used for on the fly analysis in the automotive context remains to be seen.

Salfer et al. [58] introduce a stochastic model for automatically generating attack graphs, aiming to quantify cybersecurity risks in automotive networks. This model is valuable in early development, potentially influencing critical business decisions while designing secure automotive architectures. Additionally, the authors formalize a model to assess the risk of an attacker compromising a specific asset in an in-vehicle network. This model considers the necessary budget and cost, the attacker's resources, and those essential to executing the attack.

Given that the analysis of propagation effects often relies on attack graphs, the automatic generation of these graphs based on development documents proves to be a helpful feature. The authors demonstrate that the attack generation algorithm will always terminate and can effectively handle graph cycles. The evaluation highlights that the automation of attack graph generation can be distributed across multiple computers, as the independent path extensions enable efficient MapReduce computations. Finally, the authors implement and evaluate their model, demonstrating its performance benefits over the typical Bayesian implementation of attack graphs.

Krisper et al. [59] propose the RISKEE process for risk assessment using attack graphs. This involves characterizing events in attack paths by frequency, vulnerability, and impact - representing event occurrences, attack success probability, and expected monetary loss, respectively.

After generating an attack graph, the attributes of each event must be determined. For that purpose, the authors utilize expert ratings, typically taken from a group of three to five individuals with different backgrounds. The arithmetic mean of all ratings is then calculated to express each attribute as a single value. Subsequently, the custom RISKEE propagation algorithm is employed to compute the cybersecurity risks of a given computer system.

The propagation algorithm initially extracts all possible paths from entry nodes to those defining a loss magnitude, referred to as goal nodes. For each path, the attack frequency of the entry node is propagated to all intermediate nodes, meaning that it is multiplied by the corresponding vulnerability likelihood. In addition, the magnitudes denoting the expected outcome are accumulated during this process. Finally, the cumulative risk is determined by multiplying the propagated frequency with the accumulated magnitude. This risk is then backpropagated to all nodes and edges on the path by adding it to their risk values.

According to the authors, the RISKEE process offers detailed information about potential outcomes, including confidence intervals, enabling meaningful decision-making. However, evaluating the backpropagation of risks along attack paths is yet to be conducted, leaving the benefits of this approach open to further exploration.

4. Security Requirement Analysis

The design of a secure Software-Defined Vehicle (SDV) requires a security-aware engineering process with security considerations integrated from the early development stage. Every design decision, whether related to the hardware or software architecture, must be weighed from a security perspective to establish a solid foundation for protection against cyberattacks. Only a systematic requirement analysis can effectively capture the attack surface of an SDV. Such a systematic approach typically follows a standardized process consisting of various steps, including management processes, threat and risk assessment, validation, and documentation. Furthermore, it often provides tools to help determine and assess risks from which precise security requirements can be derived. By adhering to standards, manufacturers can provide evidence to regulatory authorities of ensuring the vehicle's optimal security.

While the ISO 26262 is an internationally recognized standard for functional safety in automotive systems, the ISO/SAE 21434 [35], published in 2021, focuses explicitly on cybersecurity. These standards were designed for the automotive sector and currently dominate engineering processes. However, the significance of security in the automotive domain only became relevant with the transition from closed vehicles into interconnected SDVs. Prior to this transformation, only high-level security engineering guidelines like the SAE J3061 existed for road vehicles. Besides, security standards for other domains had already been established, like the ISA-62443 for industrial systems.

This chapter demonstrates how to perform a security requirement analysis for a SDV. We use the SDV presented in Section 1.6 as a reference vehicle. Our focus is systematically identifying threats, assessing cybersecurity risks, and mapping them to concrete technical requirements. This analysis is the first crucial step in a holistic security engineering approach for road vehicles developed from scratch. As it was mainly carried out in 2018 when the ISO/SAE 21434 did not yet exist, we utilized the ISA-62443-3-2 standard for risk assessment, Our contribution lies in the implementation of the mostly generic risk assessment from ISA-62443-3-2, aiming to identify precise requirements to protect the SDV against cybersecurity throughout its lifecycle. Moreover, we explore the applicability of the ISA-62443-3-3, which provides technical requirements, to the automotive domain and compare the standard to the novel ISO/SAE 21434.

This chapter intends to answer research question RQ1 and is the foundation for the further course of this dissertation, where we address selected security requirements.

This chapter is built upon the publication "Safety meets Security: Using IEC 62443 for a Highly Automated Road Vehicle" [1], which was presented at the *Computer Safety, Reliability, and Security* conference (SafeComp) in 2020.

4.1 Reference Architecture

The UNICARagil reference vehicles feature a distinctive zonal E/E architecture and utilize the novel Automotive Service-Oriented Architecture (ASOA) [20] for efficient software design and maintenance, along with ensuring reliable real-time communication. These vehicles, developed from the ground up, serve as a demonstration and evaluation platform for innovative concepts across various automotive domains, including automation, modularization, verification, validation, safety, and security. According to the SAE's driving automation taxonomy, these vehicles are classified as Level 4, indicating autonomous operation in known environments with the ability to reach a safe position at all times. However, human intervention may be required when operating outside these predefined environments. Given their predominantly software-driven nature, we regard the UNICARagil vehicles as ideal representatives for an SDV. They enable the addition or removal of vehicle capabilities by providing and connecting corresponding services while the E/E architecture remains consistent across all vehicle types.

As a preliminary step in the security requirement analysis, we elucidate the internal structure of the vehicles with a specific focus on the E/E and software architecture. In the following chapters, the term "reference vehicle" refers to the UNICARagil vehicle.

4.1.1 E/E Architecture

In contrast to traditional road vehicles that typically rely on domain-based architectures with numerous ECUs, the UNICARagil architecture comprises four zones, each located at one of the four vehicle corners. Each is connected to a central Ethernet network via a dedicated switch. The network follows a ring topology, ensuring communication even in the event of a link failure. The Precision Time Protocol guarantees clock synchronization among ECUs, especially critical ones affecting the driving behavior. The vehicle incorporates a total of 26 ECUs, with four primary types guiding the main event chain: sensor modules, the cerebrum, the brainstem, and dynamic modules. Not only is their terminology inspired by the human nervous system, but also their behavior, which ensures a safe state even in case of failure. Figure 4.1 illustrates the E/E architecture [60] and Table 4.1 gives an overview of the hardware and software configuration of the ECUs.

The **brainstem**, situated at the vehicle's core, operates as a fail-operational embedded real-time system for trajectory control and emergency trajectory implementation at any time. It internally employs two redundant Zynq Ultrascale+ ZU3EG chip instances, each equipped with a quad-core ARM Cortex A53 and an ARM Cortex R5 processor. The R5 processor handles real-time tasks, while the A53 processor orchestrates communication, monitors battery utilization, and manages the door control. Both processors are connected to the in-vehicle Ethernet network, utilizing the service-oriented ASOA middleware for communication.

The **spinal cord** translates low-level driving commands from the brainstem into wheel actuation, comprising four **dynamic modules**, each controlling one of the four wheels. Each wheel can be individually controlled and rotated by almost 180 degrees for lateral parking maneuvers. Each dynamic module comprises an Infineon Aurix Tricore TC27 platform with a lockstep architecture for ASIL-D software development. It can operate at temperatures between -40 °C and 150°C. They employ FreeRTOS as a lightweight OS and run the ASOA similar to the brainstem's Cortex R5 processor. The dynamic modules are redundantly connected through a FlexRay

Control Unit	Hardware	OS
Brainstem	ARM Cortex-R5	FreeRTOS
	ARM Cortex-A53	PetaLinux
Dynamic Module	Infineon Tricore Aurix TC27	FreeRTOS
Cerebrum	AMD Ryzen Threadripper 3970X, 2x Nvidia GeForce RTX 3080	Linux
Sensor Module (Perception)	AMD Ryzen Threadripper 3970X, Nvidia GeForce RTX 2080	Linux / ROS
Localization	AM335x Cortex-A8	Embedded Linux
Air Condition	Beaglebone (AM335x)	Linux
HMI / Door Control	ARM Cortex-A72 (Raspberry Pi 4)	Linux
Battery Management	ARM Cortex-A72	Linux
...	...	...

Table 4.1: The vehicle we use for evaluation consists of high-performing computing units (e.g., the cerebrum) and small embedded hardware (e.g., the spinal cord).

network for safety, ensuring communication even if the Ethernet network fails or experiences critical latency.

Four **sensor modules** [61], one in each corner, enable the vehicles to perceive their surroundings using different technologies, mitigating each other's shortcomings. Each sensor module comprises a vertical stack of one lidar, two radar, and four camera units, providing a 270° perception angle. Combining these modules offers a robust 360° view without blind spots. In addition to the sensors, each module includes an inertial measurement unit for precise localization and a trigger box for real-time synchronization. The sensor modules' main tasks are to detect traffic participants, infrastructure, and free space where the vehicle can safely navigate. Given the computational demands of sensor data processing, each module incorporates an AMD Ryzen Threadripper 3970X as the central processor and two NVIDIA GeForce RTX 3080 graphic cards. In addition to this primary perception system, the vehicles feature a secondary one as a fallback system, with platform sensors detecting near-field surroundings, ensuring a safe halt in case of severe sensor module failures.

The **cerebrum** consolidates preprocessed environment models from the sensor modules into a single-vehicle environment model. Responsible for behavior and trajectory planning, it determines a navigable corridor for the vehicle. For efficient route planning, the cerebrum communicates with a cloud that offers valuable traffic information through a collective memory. The environmental model and trajectory enable continuous self-monitoring of the vehicle's capabilities, providing input for maneuver control that selects a policy based on suitable behavior. The cerebrum possesses maneuver policies for intersections, crossing pedestrians, and regular driving modes, used to produce a trajectory covering the next five seconds.

Besides these main components, additional ECUs handle efficient battery management, provide a control interface to the passengers, and determine the precise vehicle localization. This control interface, referred to as Human-Machine Interface (HMI), is similar to the infotainment systems found in contemporary cars. However, we consistently refer to it as HMI, given the vehicles' research-focused nature that excludes entertainment services. Instead, the HMI connects the vehicle and its human occupants. It provides real-time status information about the ongoing ride and lets passengers input their destination preferences. As mentioned earlier, not only the

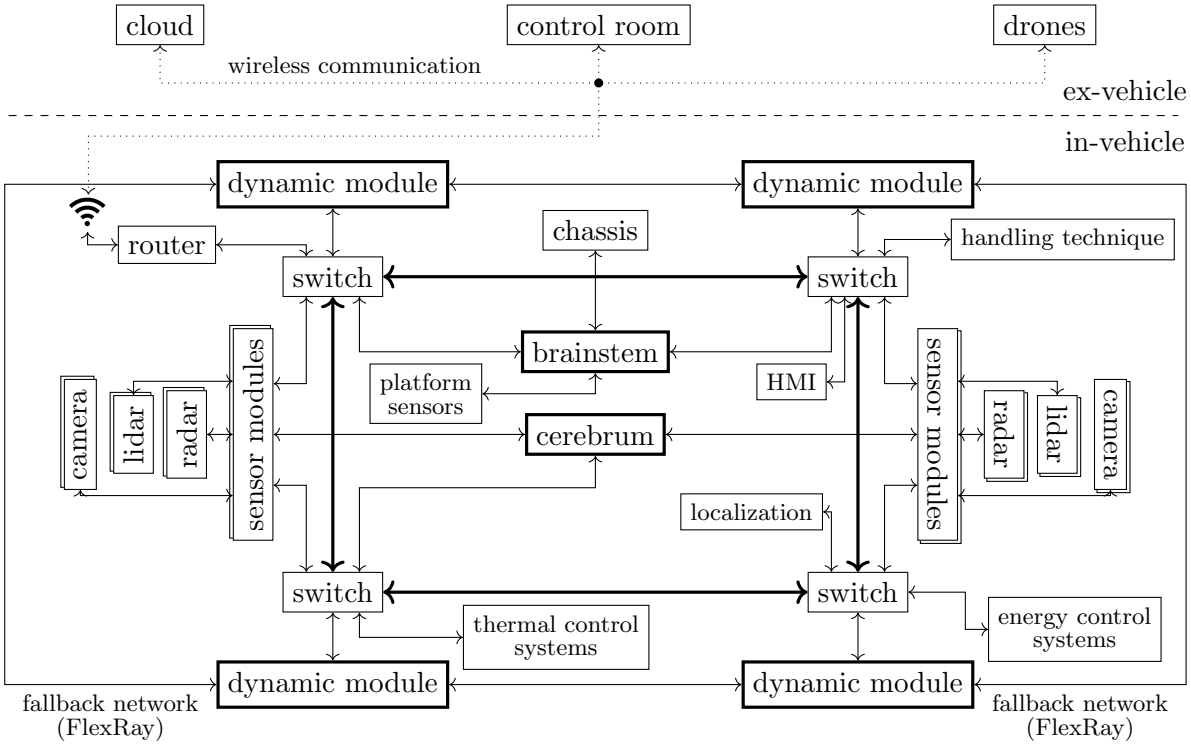

Figure 4.1: Overview of the E/E architecture of the UNICARagil vehicles

ECUs' naming is based on the human nervous system but also their behavior in exceptional situations. If a failure occurs at or before the brainstem, the spinal cord reflexively implements the trajectory, ensuring a safe halt position at any moment. This safety mechanism works even without communication on the Ethernet network since the dynamic modules are additionally wired through FlexRay.

4.1.2 Software Architecture

The vehicle's software architecture is based on the Automotive Service-Oriented Architecture (ASOA). This middleware defines a methodology for designing, developing, and maintaining functional software components, referred to as services. The primary goal is to decouple services from each other and the hardware, facilitating easy replacement and updates. Consequently, services are intentionally designed to be agnostic to the vehicle system, devoid of system-specific knowledge or dependencies on other services. As a result, central orchestration is essential to ensure proper data reception and transmission within the vehicle. For detailed information about the ASOA, please consult Section 6.1.

The ASOA is implemented across all control units, supporting resource-constrained embedded systems such as the dynamic modules or the Cortex R5 real-time processor on the brainstem. The orchestrator on the brainstem's A53 application processor establishes connections between services during vehicle bootup and whenever a vehicle's state changes.

The ASOA handles payload transmission and also appends a quality vector to transmitted data. This quality vector allows the receiver to assess the transmitted data, enhancing the vehicle's self-perception system - a critical safety feature that continuously monitors the vehicle's capabilities. For example, annotated position data can provide insight into the accuracy of the vehicle's position, enabling adaptive responses in vehicle automation.

Certain internal functions, especially those running the sensor modules and the cerebrum, leverage the Robot Operating System (ROS). The ASOA seamlessly integrates ROS services into its ecosystem, ensuring a cohesive and interoperable software environment. A central router allows ECUs to establish connections with remote entities that belong to the external infrastructure.

4.1.3 External Infrastructure

The UNICARagil vehicles are part of an automotive ecosystem, encompassing a cloud, drones, and a control room, all aiming to optimize traffic and enhance safety. The vehicles communicate with these remote components through a central router linked to the internal Ethernet network.

The **cloud** comprises three key components: the collective environment model, the collective memory, and collective behavior. Each component facilitates the vehicle's automation process and enhances passenger safety. The collective environment model aims to prevent hazardous situations by extending a single vehicle's perspective to encompass the perceived environment of surrounding vehicles. This allows, for instance, the visualization of objects hidden behind a wall. In contrast, collective memory improves automation algorithms by learning from data collected across all vehicles. The collective behavior generates safe trajectories based on previously trained models, utilizing the collective memory as the input source.

The **control room** is another essential element in the UNICARagil ecosystem. It permits human intervention when automatic maneuvering is infeasible, or automation fails to resolve a particular scenario. The control room serves as a fallback, intervening only in exceptional circumstances, enabling human operators to regain control and thereby comply with legal requirements. A typical operation occurs if a vehicle must break the law, such as crossing a sidewalk after being obstructed. In such cases, a remote human takes control of the vehicle and manually drives it using a joystick. During this process, the remote controller utilizes the vehicle's sensors to perceive its environment and reinstates automation once the vehicle is ready.

Drones play a supportive role for both the vehicle and the control room by providing additional perception data from the air.

4.2 The ISA-62443 standards

Since there was no established security requirement analysis for vehicle systems at the time of this work, we explored the applicability of the ISA-62443 standard series.

The ISA-62443 [62] constitutes a series of standards and technical reports that present a structured risk assessment and mitigation process for Industrial and Automation Control Systems (IACSs). Additionally, it offers management guidance, policies, and terminology. An IACS typically encompasses a complex system comprising various computing units, sensors, actuators, temporarily connected devices, and a human interface, collaboratively working toward a specific product outcome. The primary objective of ISA-62443-3-2 and ISA-62443-3-3 is to identify threats, assess risks, and devise protection techniques.

As depicted in Figure 4.2, the risk assessment in ISA-62443-3-2 process is detailed in consecutive steps, denoted as Zone and Conduit Requirements (ZCRs). In the first step (ZCR 1), all relevant assets of the System Under Consideration (SUC) are identified. A high-level security analysis

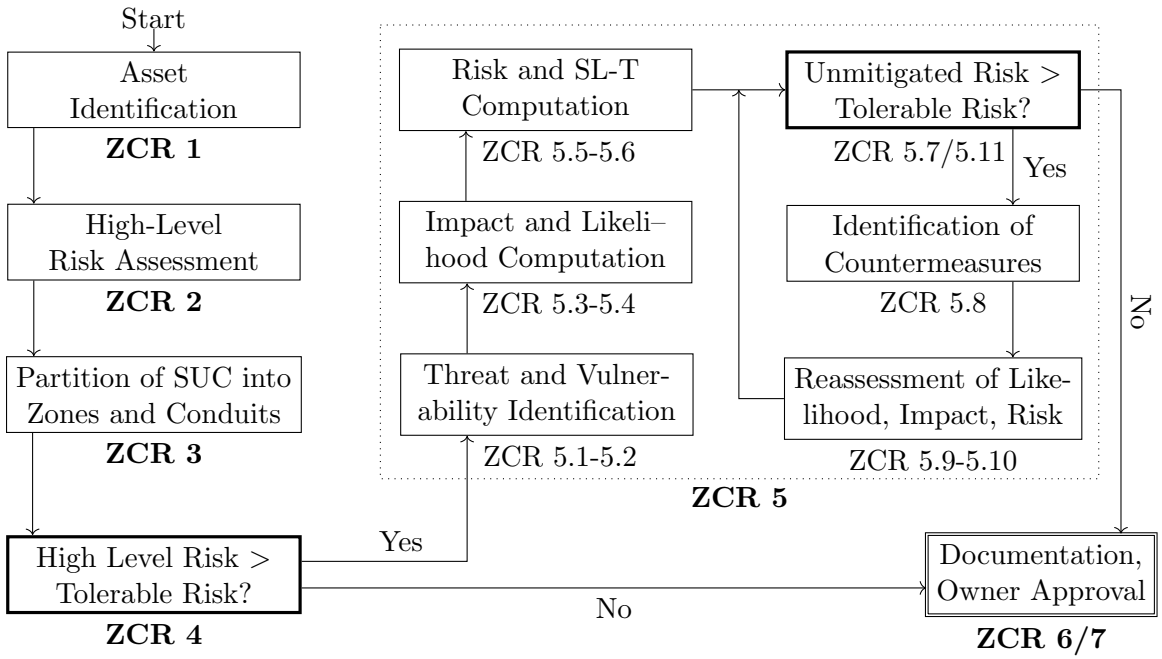

Figure 4.2: Simplified workflow of the ISA-62443-3-2

(ZCR 2) reveals the worst-case unmitigated risk on each asset and determines whether further investigation is necessary. Based on this analysis, the SUC is partitioned into zones and conduits (ZCR 3), where a zone contains related assets, for instance, in terms of functionality, localization, or safety. In contrast, a conduit is a special zone connecting two other zones. The tolerable risk (r^{tol}) of each zone is compared with the unmitigated risk r^{u} in ZCR 4. If r^{tol} surpasses r^{u}, no further action is needed; otherwise, a detailed risk assessment ensues in ZCR 5.

The primary goal of ZCR 5 is to reduce the unmitigated security risk of identified threats (T) iteratively by implementing compensating countermeasures. Threats are associated with seven Foundational Requirements (FR), namely, *Identification and Authentication Control* (IAC), *Use Control* (UC), *System Integrity* (SI), *Data Confidentiality* (DC), *Restricted Data Flow* (RDF), *Timely Response to Events* (TRE), and *Resource Availability* (RA). Section 4.3.2 presents more information about the foundational requirements.

The design of a secure IACS requires the identification of an exhaustive list of threats and exploitable vulnerabilities (ZCR 5.1-5.2). Both the impact and the likelihood of each threat (ZCR 5.3-5.4) are determined to compute the unmitigated security risk r^{u} of each threat (ZCR 5.5). Based on these results, a target security level SL-T for each zone is computed, differentiating between four levels, SL-1, SL-2, SL-3, and SL-4. While SL-0 is implicitly defined as no requirements, SL-1 requires protection against coincidental violations. SL-2, SL-3, and SL-4 cover intentional security violations with an escalating level of skills, resources, and motivation. After introducing changes to the SUC, such as implementing countermeasures, impact and likelihood are reevaluated (ZCR 5.9), ideally leading to a reduction of the residual risk (ZCR 5.10). This step, however, does not lead to a reassessment of the high-level risk. Once r^{u} of all threats falls below r^{tol}, the SUC is considered secure.

4.3 Risk Analysis using ISA-62443

We consider the reference vehicle introduced in Section 4.1 as our SUC. Although the ISA-62443 standards were not explicitly designed for automotive systems, we argue that an SDV resembles an IACS as it shares many essential properties. Specifically, the reference vehicle comprises sensors, actuators, and computing units interconnected in a network, many depending on the calculations of others. Additionally, the seven foundational requirements used to describe the SUC's security demands match the vehicle context by considering system and communication security as well as timing demands necessary in a safety-critical environment. According to the ISA-62443, the design of secure systems involves requirements affecting the organization's process, the personnel, and the technology.

Our objective is to illustrate the implementation of the generic guidelines ZCR 1-5 outlined in ISA-62443-3-2, with the ultimate aim of formulating a customized set of security requirements to ensure the secure and safe operation of the SUC. That means we concentrate on technological aspects while leaving out requirements for the organization's process and personnel. At first, we conduct a TARA using the ISA-62443-3-2 prescriptions, proposing how to implement the rather generic steps. Subsequently, we leverage our findings to define precise security requirements using ISA-62443-3-3 in conjunction with UN Regulation No. 155 [63], which encompasses additional risk mitigation proposals. For meaningful assessment, we engaged a group of eight experts from the Security Engineering Group at TU Darmstadt. The assessments provided by all experts were collected, and a mean value was calculated to obtain a final assessment.

Note that the entire security requirement analysis comprises many documents, most of which are appended to this work. In the subsequent sections, we only present excerpts for enhanced readability.

4.3.1 High-Level Risk Analysis and System Partition

ISA-62443-3-2 commences with a high-level risk analysis of zones and conduits. This allows for an initial assessment of possible security requirements, enabling the exclusion of zones that do not warrant further attention either due to their low criticality or because they are already adequately secured. In total, the high-level risk analysis comprises four ZCRs.

ZCR 1

The objective of ZCR 1 is to systematically explore the SUC, enabling a subsequent reasonable cybersecurity analysis. Initially, the SUC undergoes a partitioning process into assets, with an asset defined as a "pyhsical or logical object" necessary for the "complete automation solution".

In our context, an asset constitutes a physical functional component with the potential to impact the safe driving process, representing a concrete automation solution as denoted by ISA-62443. Therefore, we deconstruct the reference vehicle into ECUs, considering each an asset since every ECU influences the automated driving process. Besides, each external component is treated as an individual asset, given that the SUC processes data from the cloud and can be steered from the control room. Note that our focus lies in the secure design and construction of the reference vehicle, excluding detailed consideration of the external infrastructure. Although we acknowledge the complexity of the cloud, drones, and the control room, which would require a

dedicated security requirement analysis, we treat them as monolithic external communication nodes in this study.

In the process of identifying assets, we also have to recognize access points to the SUC, as they serve as potential gateways for attackers to infiltrate the system. Our experts identified a total of 19 asset types, notably including the sensor modules, the cerebrum, the brainstem, and the dynamic modules. The term "type" refers to the fact that some ECUs exist multiple times but are counted as a single asset. For instance, four dynamic and four sensor modules are inside the SUC. Since they are identically constructed, we treat them as two separate assets. Table A.2 provides a complete list of the identified assets. Additionally, the SUC possesses an access point through the router, as all incoming and outgoing communication flows must cross this router.

ZCR 2

After the SUC has been divided into assets, a high-level cybersecurity risk assessment is conducted in ZCR 2 to determine an initial security target level for each asset a_i. According to ISA-62443-3-2, evaluating both the high-level likelihood $L_{a_i}^{\mathrm{HL}}$ and the high-level impact $I_{a_i}^{\mathrm{HL}}$ of a potential attack on a_i is necessary. However, this standard does not prescribe a specific method for this assessment. To address this, we propose employing a multi-criteria decision-making process, which is especially useful when a decision involves multiple criteria rather than a single one.

This approach allows each criterion to be weighted and establishes a hierarchy among them. Technically, we represent both the high-level likelihood and impact as vectors $\vec{L}_{a_i}^{\mathrm{HL}} = (L_1\ L_2\ ...\ L_n)^\intercal$ and $\vec{I}_{a_i}^{\mathrm{HL}} = (I_1\ I_2\ ...\ I_m)^\intercal$, respectively. Each vector field signifies a specific criterion, initially ranked by experts and then weighted based on their criticality. After scoring each criterion, we normalize $L_{a_i}^{\mathrm{HL}}$ and $I_{a_i}^{\mathrm{HL}}$, followed by a multiplication with precomputed weight matrices, denoted as $L_{a_i}^{\mathrm{HL}} = \vec{L}_{a_i}^{\mathrm{HL}} \cdot \vec{W}_{\mathrm{L}}$, respectively $I_{a_i}^{\mathrm{HL}} = \vec{I}_{a_i}^{\mathrm{HL}} \cdot \vec{W}_{\mathrm{I}}$, where $\cdot$ is the dot product. Normalization ensures that a score of $L_{a_i}^{\mathrm{HL}}$ =1 indicates the highest possible likelihood, while $I_{a_i}^{\mathrm{HL}}$ =1 stands for the worst-case impact. Finally, the high-level risk $r_{a_i}^{\mathrm{HL}} = (I_{a_i}^{\mathrm{HL}}, L_{a_i}^{\mathrm{HL}},)$ is mapped to a risk class using the weighted normalized decision matrix in Table 4.6.

Impact Assessment We evaluate the anticipated impact of a cyberattack on a given asset a_i across four criteria: *Passenger Safety* (PS), *Financial Loss* (FL), *Operational Restrictions* (OR), and *Privacy* (P). Consequently, we express $\vec{L}_{a_i}^{\mathrm{HL}}$ = (PS FL OL P)$^\intercal$ as a vector of four impact criteria. Our experts independently score each criterion using a set of exclusive parameters, denoted as $\mathcal{P}$. Table A.1 describes the notion of each parameter and the corresponding quantified value.

- **Passenger Safety (PS)**: Safety considerations include the well-being of passengers and those near the vehicle. Parameters range from *no injuries* to *fatal*, associated with integer values from 1 to 4. That means, $\mathcal{P}_{\mathrm{PS}} = \{$*fatal*, *seriously injured*, *slightly injured*, *no injuries*$\}$.
- **Financial Loss (FL)**: This criterion addresses the monetary consequences for the manufacturer or vehicle operator in the event of a successful cyberattack. Parameters range from "severe" to "negligible", with severity based on the Value of Statistical Life (VSL) [64] assumed to be $10 million.

– **Operational Restrictions (OR)**: Operational limitations resulting from a cyberattack are expressed in terms of "massive", "high", "medium", and "low". A *massive* limitation occurs if all traffic comes to a halt. *High* limitations lead to traffic jams in a designated area, while *medium* constraints occur when a vehicle can only operate at reduced speed. Finally, *low* limitations result from hijacking non-critical assets such as the chassis.

– **Privacy (P)**: Privacy criteria range from severe to negligible, reflecting the degree of sensitive data leakage following a successful attack. A *severe* privacy leakage occurs if the leaked data reveals the passenger's identity and can be directly linked to him. In contrast, *major* privacy leakage requires low computational power to link stolen data to the passenger, while a *moderate* leakage necessitates data in large quantities to reveal personal information. A *negligable* privacy leakage happens when the data does not allow any conclusions to be drawn about the passenger.

In this analysis, the focus is solely on the functional description of asset a_i, excluding propagating side effects. Otherwise, all assets would receive the highest impact score, which could result in over-engineering. Transitive attacks are covered by conduits in later steps.

Table A.2 presents unweighted impact ratings for each asset during a successful cyberattack. For instance, the corruption of a sensor module may result in severe privacy issues due to its use of cameras, radars, and lidar for environmental perception. However, the impact on passenger safety is expected to be moderate because a corrupted sensor module does not directly influence driving dynamics such that the vehicle can still come to a safe halt. For the same reason, the financial loss is likewise expected to be moderate in the worst case. We expect medium operational restrictions as the vehicle may only drive at reduced speed, causing traffic jams.

No	**Asset a_i**	**PS**	**FL**	**OR**	**P**
1	sensor module	moderate	moderate	medium	severe
2	cerebrum	moderate	moderate	medium	moderate
3	brainstem	moderate	moderate	medium	moderate
...	...	...	...	...	...

Table 4.2: For each asset, the potential impact of a successful attack is evaluated, quantified, and presented as a four-dimensional vector encompassing Passenger Safety (PS), Financial Loss (FL), Operational Restrictions (OR), and Privacy (P). A detailed overview of the impact ratings is presented in Table A.2.

Impact Weighting. To express the overall impact of a cyberattack on a given asset, a single value is calculated, incorporating individual criteria scores. As mentioned earlier, these criteria undergo weighting in a multi-criteria decision-making process, emphasizing their relevance. In our context, passenger safety is the highest priority, followed by financial loss and operational restrictions, which are considered equally relevant. Privacy has the lowest priority, indicating a preference for accepting a serious privacy incident over a severe injury.

We suggest following a Analytic Hierarchy Process (AHP) to derive meaningful weights for each criterion. The Analytic Hierarchy Process (AHP), developed by Thomas L. Saaty in the 1980s [65], enables complex decision analysis by breaking down decisions into simpler options and performing pairwise comparisons. Table 4.3 shows the pairwise comparison of our impact

options (PS, FL, OR, P). Passenger safety is three times more relevant than privacy and twice as important as financial and operational consequences.

	PS	**FL**	**OR**	**P**
PS	1	2	3	4
FL	$\frac{1}{2}$	1	1	2
OR	$\frac{1}{2}$	1	1	2
P	$\frac{1}{3}$	$\frac{1}{2}$	$\frac{1}{2}$	1

Table 4.3: The pairwise comparison matrix, an essential part of the Analytic Hierarchy Process (AHP) method, shows the relevance between all possible impact criteria.

According to the AHP method, a given criterion's final weight is the corresponding row's normalized geometric mean, leading to $\vec{W}_{\mathrm{L}} = (0.423147, 0.2273505, 0.2273505, 0.122152)$ as outlined in Table 4.4.

Criterion	**Weight**
Passenger Safety	0.42315
Financial Loss	0.22735
Operational Restrictions	0.22735
Privacy	0.12215

Table 4.4: Weighted impact criteria used in the AHP.

Note that the AHP method offers a consistency check to ensure logical weightings without contradictions. This is the case if the quotient of the consistency index and the so-called random consistency index are smaller than 0.1. In our case, the consistency ratio of our pairwise comparisons is 0.003836 [66], confirming the consistency of the weights.

Impact Computation. Having assessed the impact categories (Table 4.3) and determined the impact weights (Table 4.4), an impact value for each asset is computed through $L_{a_i}^{\mathrm{HL}} = \vec{L}_{a_i}^{\mathrm{HL}} \cdot \vec{W}_{\mathrm{L}}$ after normalizing $\vec{L}_{a_i}^{\mathrm{HL}}$. Table A.2 shows a complete overview of the impact values of all assets. For example, the expected impact resulting from a corrupted sensor module is computed as follows:

$$\text{Impact} = \frac{2}{4} \cdot 0.42315 + \frac{2}{4} \cdot 0.22735 + \frac{3}{4} \cdot 0.22735 + \frac{4}{4} \cdot 012215 = \mathbf{0.61791} \tag{4.1}$$

Likelihood Assessment The likelihood parameter represents an attacker's difficulty in compromising a given asset. Recall that we consider standalone devices such as ECUs or external entities like the control room as assets. Assets with open interfaces generally pose an easier target than those physically isolated without connectivity. We have identified six rating criteria to capture this, all providing an adversary with options to gain unauthorized access.

These criteria are represented as Boolean fields within a six-dimensional vector, denoted as $\vec{L}_{a_i}^{\mathrm{HL}} = (\mathrm{IC\ WI\ U\ EI\ B\ TP})^{\intercal}$. This vector determines for each asset a_i whether the following conditions are met: the ability to establish an *Internet Connection* (IC), the presence of *Wireless Interface* (WI), the asset's *Updatability* (U) nature, the existence of *External Interfaces* (EI) such

as OBD2, USB, direct connection to the in-vehicle *Bus* (B), and whether a_i deploys software by a *Third Party* (TP).

For instance, the vector $\vec{L}_{a_i}^{\mathrm{HL}} = (1\ 0\ 1\ 0\ 1\ 0)^\intercal$ denotes an updatable asset connected to the in-vehicle bus with the capability to establish an Internet connection. Table 4.5 shows a selection of rated assets according to the previously identified likelihood criteria. The criteria order implicitly

No	**Asset** a_i	**Likelihood Criteria**						**Likelihood** $L_{a_i}^{\mathrm{HL}}$
		IC	WC	U	EI	B	TP	
1	sensor module	1	0	1	1	1	0	0.71
5	lidar	0	0	0	0	1	1	0.11
8	hmi	1	1	1	1	1	0	0.96
16	control room	1	1	1	1	0	0	0.89

Table 4.5: Each likelihood criterion is assessed and weighted, resulting in a likelihood value.

reflects their relevance in terms of their potential to facilitate an attack. That means an asset connected to the Internet is more susceptible to be attacked than an asset without an Internet connection but with a USB interface, as the attacker would need to be physically present.

We again apply an AHP to mathematically express each criterion's relevance, resulting in distinct weights. This again involves a pairwise comparison of all likelihood criteria, followed by normalization, resulting in $\vec{W}_{\mathrm{L}} = (0.38\ 0.25\ 0.16\ 0.10\ 0.06\ 0.04)^\intercal$. Finally, an asset's high-level likelihood $L_{a_i}^{\mathrm{HL}}$ is obtained by computing $\vec{L}_{a_i}^{\mathrm{HL}} \cdot \vec{W}_{\mathrm{L}}$. Table A.2 presents our assessment results and the corresponding likelihood values.

Risk Assessment After assessing the attack impact and likelihood for each asset a_i, we can derive a risk value. ISA-62443-3-2 suggests using a risk matrix that relates the impact $I_{a_i}^{\mathrm{HL}}$ to the likelihood $L_{a_i}^{\mathrm{HL}}$. However, as in the previous steps, no precise instructions for implementation are given by the standard, so we divide the spectrum of likelihood and impact into the five classes "low", "medium", "major", "high", and "very high". Such an even division is possible because the previously calculated impact and likelihood values are normalized. Table 4.6 shows the resulting risk matrix, which we now use to determine a risk value for each asset a_i. This

			Likelihood $L_{a_i}^{\mathrm{HL}}$				
			negligble	**very low**	**low**	**medium**	**high**
			0	(0, 0.25]	(0.25, 0.5]	(0.5, 0.75]	(0.75, 1]
Impact $I_{a_i}^{\mathrm{HL}}$	**negligible**	[0, 0.25]	low	low	low	low	low
	moderate	(0.25, 0.5]	low	low	medium	medium	major
	major	(0.5, 0.75]	low	medium	major	high	very high
	severe	(0.75, 1]	low	major	high	very high	very high

Table 4.6: Risk matrix with grey cells indicating a tolerable risk.

calculated risk value is the result of ZCR 2. It is used in subsequent steps to partition the SUC and to answer whether a detailed security requirement analysis is necessary. Again, Table A.2 provides a detailed overview of the risk values for all assets.

Note that we intentionally added a likelihood of zero in the risk matrix to express the successful mitigation of a threat in later steps. With this column in the risk matrix, threats with a severe impact could be considered mitigated since security measures will only significantly lower the likelihood of a threat. At the same time, the safety consequences remain the same. Consequently, the objective is to make such attacks impossible by applying sound technical means. In such a situation, we assign a zero likelihood, resulting in a tolerable risk.

The risk analysis confirms that assets exerting direct or indirect influence on driving dynamics or possessing numerous external interfaces carry a significant cybersecurity risk. For instance, the dynamic modules responsible for propelling the reference vehicle constitute a noteworthy cybersecurity risk, thereby warranting a classification of a *very high* risk. Conversely, sensor modules, tasked with perceiving the vehicle's surroundings and likewise crucial for safe driving, have a *high* cybersecurity risk. The sensor modules are remotely updatable and indirectly impact driving behavior through environmental perception. However, an attacker would need to simultaneously seize control of most sensor modules to compromise the vehicle's ability to perceive its surroundings correctly. Additionally, the vehicle can even reach a safe position without sensor modules. Therefore, these ECUs are not categorized in the highest risk class. The vehicle's nearfield sensors have a *medium* cybersecurity risk, as neither major safety consequences are expected nor a significant likelihood for compromise is given. About external assets, the drone likewise poses a *medium* risk since it only provides additional environmental information on a high level. At the same time, the control room belongs to the *very high* risk category since an attacker may gain access not only to one vehicle but to an entire vehicle fleet.

ZCR 3

The objective of ZCR 3 is to partition the SUC into zones and conduits to simplify the ensuing detailed security requirement analysis of the overall system. The standard defines a zone as a grouping of assets based on functional, logical, or physical relationships. Physical zones cluster nearby assets, while functional zones comprise assets with similar or related functionality. Nested zones are also possible, i.e., one zone can encompass another. Additionally, virtual zones categorize assets based on specific properties, such as common security requirements.

In our context, virtual zones serve to group assets sharing identical cybersecurity risks. Instead of creating a dedicated zone for each risk class, we incorporate other factors for a meaningful partition of the SUC, as displayed in Figure 4.3. First, we separate vehicle-internal and ex-

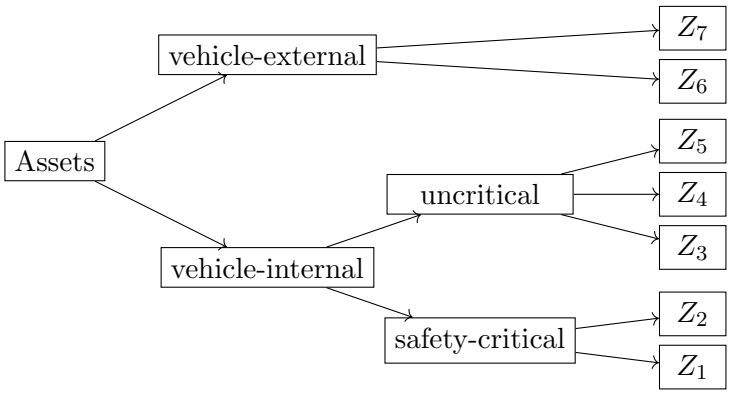

Figure 4.3: For the partitioning of the SUC into zones and conduits, we considered not only the security risk of the assets but also whether they are located inside or outside the vehicle and whether they execute safety-critical tasks.

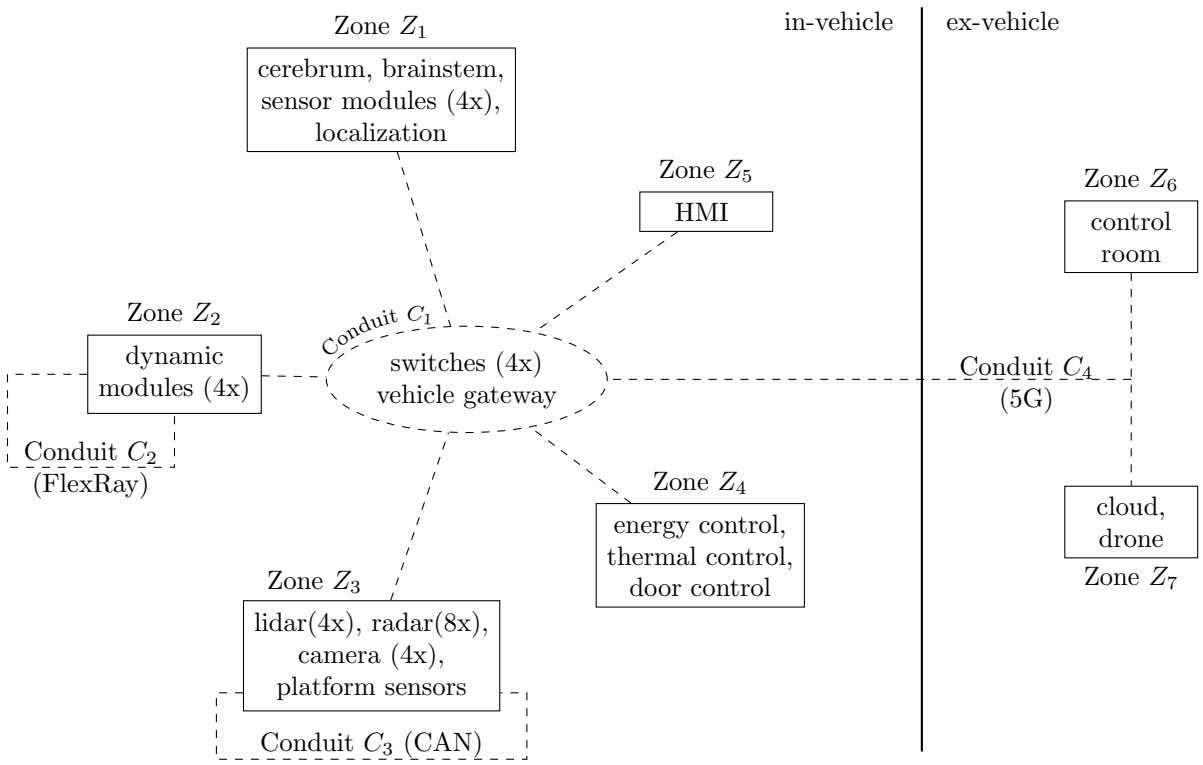

Figure 4.4: The SUC is partioned into seven zones and four conduits. Conduits C_2 and C_3 are nested in the zones Z_2 and Z_3 because they only allow for communication within these zones. Conduit C_1 represents the main Ethernet-based in-vehicle bus.

ternal assets; then, we differentiate between their safety relevance by considering all assets as safety-critical whose impact rating is either *major* or *severe*. Finally, we consider the functional characteristics of individual assets within vehicles, such as the four dynamic modules forming a unit known as the spinal cord. Despite their distribution across all four corners of the vehicle, we assign them to the same zone due to their functional relationship. In total, the SUC is divided into seven zones $Z_1, ..., Z_7$ as illustrated in Figure 4.4. Figure 4.3 illustrates the relationship between these zones, while Table A.2 lists the corresponding zone for each asset.

Zone Z_1 comprises the ECUs responsible for environment perception, route planning, and control, all having a *high* cybersecurity risk. As earlier mentioned, the dynamic modules belong to a dedicated zone Z_2 because they bear a *very high* cybersecurity risk and directly control the wheels. Zone Z_3 houses lidar, radar, camera, and the nearfield platform sensors, either with a low or medium cybersecurity risk. Severe consequences are only anticipated if an attacker gains control over the majority of these sensors. In contrast, zone Z_4 contains low-level, safety-critical assets such as the energy and door control system, but without directly impacting driving dynamics. Zone Z_5 accommodates the HMI module. Although its cybersecurity risk is akin to that of Zone Z_4, we deliberately assign it to a dedicated zone as it is the sole ECU with a user interface. Processing user input and potentially managing different user accounts, we anticipate distinct security requirements for this zone.

Finally, Z_6 and Z_7 incorporate the vehicle-external assets. The control room, capable of directly controlling any vehicle, is designated with a *very high* cybersecurity risk and is placed in the dedicated zone Z_6. As for the drone and cloud, we opted to position them in Z_7 as they serve a supportive role without direct influence on driving behavior. Their *high* security risk stems from their update capability and interfaces, which expand the attack surface.

Besides the zones, we identified four conduits $C_1, ..., C_4$. A conduit is a particular type of zone that enables communication between zones. C_1 encompasses the in-vehicle Ethernet network, four switches, and the router. C_2 denotes the FlexRay network providing a fallback level between the dynamic modules, while the CAN network linking the platform nearfield sensors to the brainstem is placed in C_3. Both C_2 and C_3 are nested conduits inside zone Z_2 and Z_3, respectively, since they only allow for communication within these zones. Finally, C_4 represents the wireless communication between the SUC and remote assets, such as the control room and the cloud.

ZCR 4

As part of ZCR 4, we must decide which zones require a detailed security assessment. Previously, we assigned a high-level cybersecurity risk $r_{a_i}^{\mathrm{HL}}$ to each asset a_i and partitioned the SUC into zones and conduits. The risk $r_{a_i}^{\mathrm{HL}}$ assists us in identifying the need for extended security analysis, as mandated by ISA-62443-3-2.

For that purpose, we compare each asset's $r_{a_i}^{\mathrm{HL}}$ with the zone's maximum tolerable risk $r_{Z_i}^{\mathrm{tol}}$. We only exclude a zone from further consideration if all its assets have a risk value below the tolerable risk. That way, we eventually define security requirements solely for zones needing protection, preventing over-engineering in subsequent phases.

The ISA-62443-3-2 does not specify a methodology for determining $r_{Z_i}^{\mathrm{tol}}$. Therefore, we elaborated on this, considering only the grayed cells in the risk matrix presented in Table 4.6 as acceptable risks. In other words, r^{tol}= *low* holds, meaning that zones with a *low* security risk do not require further attention if they are not connected to critical zones through a conduit. Notably, only zone Z_3 houses assets with *low* and *medium* cybersecurity risks. However, Z_3 is connected to C_1 as its assets can communicate via the in-vehicle Ethernet network. Consequently, security incidents may propagate through C_1 into Z_1 or Z_2, containing safety-critical assets. Therefore, Z_3 must be included in subsequent security analysis steps. As all other zones exhibit either *high* or *very high* risks, they are also part of the following detailed security analysis.

4.3.2 Detailed Cybersecurity Risk Assessment

ZCR 5 outlines the steps of a detailed cybersecurity risk assessment, which becomes mandatory when a zone's previously determined high-level risk exceeds the maximum tolerable risk $r_{Z_i}^{\mathrm{tol}}$. This detailed analysis consists of thirteen substeps (ZCR 5.1-ZCR 5.13). The term "detailed" indicates that impact, likelihood, and risk values are not determined at the asset granularity but are instead used to assess individual threats.

The substeps aim to identify and assess threats at a fine-grained level, establish a target security level for each zone, and eventually mitigate the unmitigated risk $r_{Z_i}^{\mathrm{u}}$ by applying compensating measures. In that context, security level quantifies security demands arising from risks, where a risk results from a threat on a given asset combined with at least one vulnerability. As mentioned earlier, the standard uses seven foundational requirements to determine security levels in response to the identified threats. A threat is a specific attack compromising a system's confidentiality, integrity, or availability. Compensating measures encompass specific security requirements that affect both software and hardware. These measures contribute to the design of a secure SUC in reference to a predefined set of threats and vulnerabilities, eventually lowering a zone's $r_{Z_i}^{\mathrm{u}}$. Consequently, the achieved security level SL-A_{Z_i} advances towards the target security level SL-T_{Z_i}.

The ISA-62443-3-2 permits choosing an appropriate risk assessment methodology in ZCR 5. However, it emphasizes the importance of maintaining consistency with the risk ranking scale established in earlier stages. The assessments can only yield meaningful and coherent results by adhering to this criterion.

Foundational Requirements

The ISA-62443-1-1 standard defines seven categories, termed foundational requirements, forming the basis for subsequent security assessments:

1. Identification and authentication control (IAC)
2. Use control (UC)
3. System Integrity (SI)
4. Data confidentiality (DC)
5. Restricted data flow (RDF)
6. Timely response to events (TRE)
7. Resource availability (RA)

These foundational requirements FR are used to characterize a system's security properties and determine security levels in ISA-62443-3-2. That way, the vague term "security" gets a concrete meaning through the foundational requirements. They enable the measurement and quantification of the SUC's security, particularly by assessing each cybersecurity threat in terms of the foundation requirements it violates. Ultimately, we manifest a security level as a seven-dimensional vector of foundational requirements, expressing the security demands for each zone. This approach facilitates deriving specific compensating technologies to mitigate previously identified threats.

Alternatively, the CIA triad offers a method for assessing security threats by distinguishing between confidentiality, integrity, and availability. The foundational requirements offer a more nuanced perspective than the CIA triad, especially concerning system integrity and resource availability. In our context, these requirements prove advantageous as they simplify the consideration of communication and system security. Moreover, the TRE requirement facilitates the description of attacks attempting to increase latency in safety-critical environments - a scenario that demands attention, especially in road vehicles.

Threat Modeling

ZCR 5.1 prescribes to compile a list of threats that could negatively affect assets. The biggest challenge lies in developing a comprehensive and realistic list of threats T, as all subsequent analysis steps depend on that list. Ultimately, the SUC will be secured against these threats, making an incomplete threat list an opportunity for attackers to infiltrate the system.

During threat identification, we face two core problems: Ensuring completeness for T is a difficult, if not impossible, task despite various supportive techniques, such as CIA, STRIDE, and Threat Trees [67]. By relying on our expert group, we claim to have diverse views on the SUC and to obtain a reasonable number of threats. Additionally, we acknowledge the work by Petit and Shladover [68], who identified potential attack surfaces on road vehicles, inspiring our threat identification.

Moreover, a collaborative threat identification process requires a common notion of the *granularity* level of a threat. For instance, t_1 = *"The attacker triggers the vehicle brakes."* and t_2 = *"A network man-in-the-middle attacker injects forged braking commands."* are both potential threats with the same outcome. However, t_1 is articulated on a purely functional level, while t_2 already addresses *one* potential attack vector. The author of t_1 might perceive the SUC at a coarser granularity level, potentially missing attack vectors, as more than one vector can lead to the same outcome.

To address the challenge of a common notion, we propose a three-round threat identification process in ZCR 5.1, ensuring scalable threat management with threats at a comparable granularity level. In the first round, we identify *top-level* threats on a purely functional level, focusing on *what* consequences are possible. In the second round, each top-level threat is decomposed into intermediate threats, taking into account *how* they can be realized. An intermediate threat always refers to at least one specific asset, hence providing a precise attack vector as required by the ISA-62443-3-2 standard. Since an attack vector can be used to realize more than one attack, an intermediate threat may appear multiple times. For instance, threats $t_{0\text{-}2}$ and $t_{2\text{-}2}$ in Table A.3 are identical and are thus treated equally in succeeding steps.

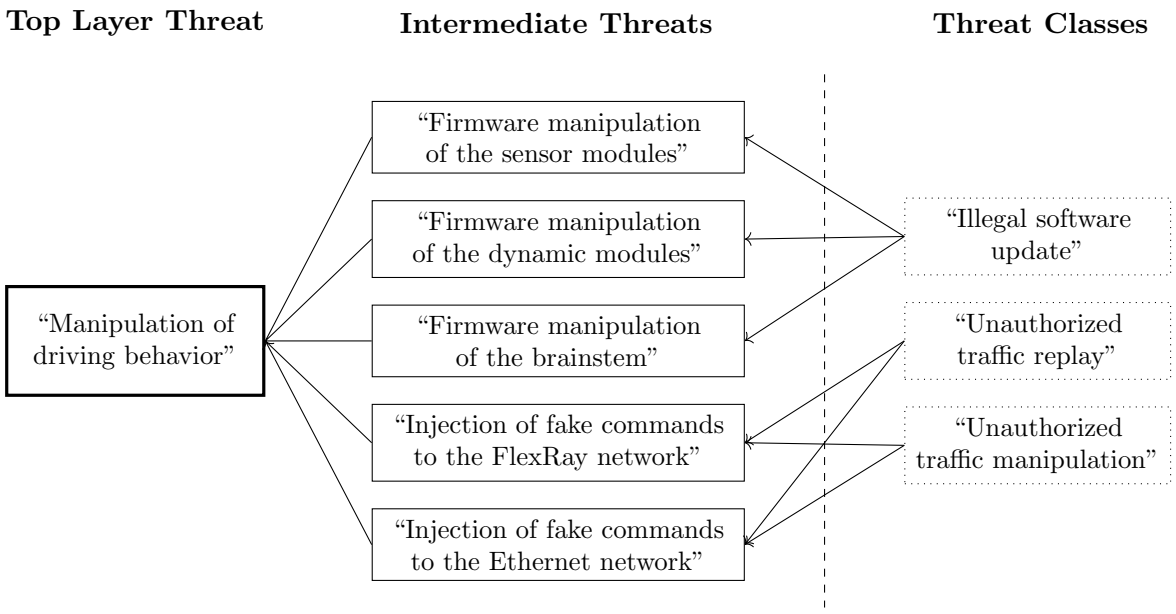

Figure 4.5: A top-level threat describes adverse functional actions composed of potentially several intermediate threats. An intermediate threat represents *how* a specific asset is compromised and may reoccur during the threat analysis. Intermediate threats are generalized in threat classes.

In the third and final round, intermediate threats are further organized into threat classes. Threat classes are generalized threats considered essential ingredients to implement an attack. For instance, the threat class of "unauthorized traffic manipulation" may impact all in-vehicle networks, contingent on the attacker's capabilities and objectives. Given that our SUC encompasses three distinct networks - Ethernet, CAN, and FlexRay - this threat is replicated three times, eventually resulting in tailored security requirements for each bus technology. Similarly, the threat of "illegal software update" pertains to all ECUs and, therefore, must be considered 26 times. We argue that duplicating threats and their separate consideration is crucial since different ECUs may possess distinct update mechanisms, demanding varying resources.

In later steps, threat classes help map intermediate threats to foundational requirements. They are divided into two categories for a better organization: communication and hardware. The communication category consolidates threats involving any adverse behavior on the network, while the hardware category encompasses threats affecting ECUs, such as software manipulation or key leakage. These categories later translate into different attacker models, each necessitating specific capabilities to implement a threat. Figure 4.5 illustrates the relationship between the threat types.

Threat Assessment In ZCR 5.1, we identified a total of 48 intermediate threats and categorized them into 9 threat classes. The identified threat classes are outlined in Table 4.7, and a comprehensive overview of the threats can be found in Table A.3 in the appendix. Impact

Category	**Threat Class**	**Abbreviation**	**FRs**
Communication	Unauthorized traffic manipulation	TC1	SI
	Unauthorized traffic replay	TC2	SI
	Eavesdropping of privacy sensitive data	TC3	DC
	Privilege escalation	TC4	IAC
	Denial-of-Service	TC5	RA
Hardware	Illegal software update	TC6	SI
	Leakage of cryptographic secrets	TC7	IAC, SI, DC
	Unauthorized connection to interfaces	TC8	SI
	Identity theft	TC9	IAC

Table 4.7: A threat class is a generalized intermediate threat. Each one is uniquely mapped to a selection of foundational requirements

(ZCR 5.3) and unmitigated likelihood (ZCR 5.4) assessments were conducted for each intermediate threat, resulting in the determination of a cybersecurity risk (ZCR 5.5). We applied almost the same metrics utilized in the high-level security analysis conducted in ZCR 2 throughout these steps, however, enriching the likelihood assessment with additional aspects. That way, we expect more realistic assessments.

More precisely, we suggest a *cascading parameter approach* that assesses likelihood based on additional factors such as vulnerabilities and attacker's capabilities. Specifically, the likelihood of a successful threat t_i is determined by the necessary capabilities for its execution and exploitable vulnerabilities (ZCR 5.2). We acknowledge the prior proposal of integrating the attacker's capabilities into threat likelihood in [69].

A vulnerability refers to a technical flaw that, if exploited, allows attackers to breach the system. Databases like the CVE system contain known software vulnerabilities that can be referenced for each ECU. Each vulnerability is assigned a value from $\mathcal{V} = \{severe, medium, negligible\}$. For example, a severe vulnerability might involve a broken cryptographic protocol or a zero-day exploit, while a medium one may require user privileges. A negligible vulnerability is primarily theoretical, such as quantum attacks.

For simplicity and due to resource limitations during the risk analysis, we assume a "medium" vulnerability for each ECU instead of researching real-existing vulnerabilities as required in ZCR 5.2. However, we point out that the systematic recording of vulnerabilities remains pertinent for commercial use cases.

In relation to the capabilities of the attacker, we utilize three factors, akin to the HEAVENS project [39]: *Experience* ($\mathcal{E}$), *Knowledge* ($\mathcal{K}$), and *Resources* ($\mathcal{R}$), denoted collectively as $\mathcal{AC} = \mathcal{E} \times \mathcal{K} \times \mathcal{R}$. Concerning expertise, we categorize individuals into three levels: *layman*, *proficient*, and *expert*. Regarding knowledge, a three-tier distinction is made: *public*, *restricted*, and *sensitive*. In terms of resources, we differentiate among *standard*, *specialized*, and *bespoke*. These potential characteristics are also drawn from the HEAVENS project. As illustrated in Table A.4, each factor is independently assessed for all threats.

Security Levels

In ZCR 5.6, we establish a target security level, denoted as SL-T$_{Z_i}$, for each zone $Z_i \in \mathsf{ZC}$. This level indicates the envisioned degree of protection for a particular zone, typically involving aspects such as access control, confidentiality, authentication methods, and software integrity. The ISA-62443-3-2 standard does not prescribe the method for computing SL-T but merely recommends representing it either as a scalar or a vector. While a scalar value is simpler to determine and minimizes verification efforts, it lacks the capacity for fine-grained resolution across various security aspects. For instance, a zone may necessitate strong authenticity protection and thus be assigned a high-security level, while other security considerations, such as confidentiality or access control, may be less critical. Encoding such nuanced information into a single scalar is challenging and may lead to over-engineering.

Therefore, we express the target security level of a zone $Z_i \in \mathsf{ZC}$ as a seven-dimensional vector, with each field representing a foundational requirement. This approach enables the individual weighting of each requirement, yielding a more meaningful and subtle representation. In other words, for each zone, we assign a security level to each foundational requirement, thereby expressing to what extent it is affected in Z_i. Recall that ISA-62443-1-1 defines five security levels (SL-0 - SL-4). SL-0 indicates the unnecessity of security protection, while SL-1 demands protection against coincidental violation. SL-2, SL-3, and SL-4 have in common to protect against intentional violation with increasing resources, skills, and motivation.

To derive a meaningful SL-T$_{Z_i}$ for zone Z_i, we begin by identifying the foundational requirements influenced by threats within Z_i. This is possible because we explore each threat's impact on specific zones (c.f., Table A.3), coupled with the additional mapping of each threat to a threat class. Since each threat class is distinctly linked to a set of foundational requirements $\mathsf{FR}' \in \mathsf{FR}$, we can associate the corresponding threat with FR'.

For instance, threat $t_{0,1}$ describes the firmware corruption of a dynamic module through a fake update, thereby associating it with zone Z_2. $t_{0,1}$ is mapped to threat class TC6, which in turn refers to the foundational requirement "System Integrity". Since $t_{0,1}$ lies in zone Z_2, this zone requires protection concerning system integrity.

Eventually, we determine a security level for each field of SL-T$_{Z_i}$ depending on whether the foundational requirement is affected in zone Z_i.

Risk Comparison

ZCR 5.7 mandates the comparison each threat's unmitigated risk $r_{t_i}^{\mathrm{u}}$ with the tolerable risk threshold r^{tol}. Only if $r_{t_i}^{\mathrm{u}}$ exceeds r^{tol} should the threat t_i be taken into account in subsequent steps for identifying compensating measures.

Zone/Conduit	IAC	UC	SI	DC	RDF	TRE	RA
Z_1	SL-4	SL-4	SL-4	SL-4	SL-0	SL-4	SL-0
Z_2	SL-4	SL-0	SL-4	SL-0	SL-0	SL-4	SL-0
Z_3	SL-4	SL-0	SL-4	SL-4	SL-0	SL-4	SL-0
Z_4	SL-4	SL-0	SL-4	SL-0	SL-0	SL-4	SL-0
Z_5	SL-4	SL-4	SL-4	SL-0	SL-0	SL-3	SL-0
Z_6	SL-4	SL-4	SL-4	SL-0	SL-0	SL-4	SL-0
Z_7	SL-3	SL-0	SL-3	SL-0	SL-0	SL-3	SL-0
C_1	SL-0	SL-0	SL-4	SL-4	SL-4	SL-0	SL-4
C_2	SL-0	SL-0	SL-4	SL-0	SL-4	SL-0	SL-4
C_3	SL-0	SL-0	SL-4	SL-0	SL-4	SL-0	SL-4
C_4	SL-0	SL-0	SL-3	SL-3	SL-3	SL-0	SL-3

Table 4.8: A zone's security target level is a seven-dimensional vector with each field representing a foundational requirement.

Similar to step ZCR 4, where we compared an asset's high-level risk with the corresponding zone's tolerable risk $r^{\text{tol}}_{Z_i}$, we designate the grayed cells in our risk matrix as tolerable (c.f., Table 4.6). Our analysis shows that $r^{\text{u}}_{t_i} > r^{\text{tol}}$ holds for each threat. Consequently, all threats must be considered in the ensuing steps. This finding is unsurprising, as the SUC has not yet been secured and cannot effectively defend against attackers. SUC.

To express the actual security level the SUC provides, the ISA-62443-3-2 uses SL-A$_{Z_i}$. Since the SUC is still in the design phase, we initially did not assume any security measures, resulting in SL-A=(SL-0 SL-0 SL-0 SL-0 SL-0 SL-0 SL-0) for all zones. The SL-T$_{Z_i}$ from Table 4.8 help us identify appropriate countermeasures, which decrease the residual risk of each threat by fortifying the SUC against cyberattacks and meanwhile approximate SL-A to SL-T.

4.3.3 Threat Mitigation

The final phase of the detailed security analysis revolves around identifying compensating measures that lower the unmitigated threat risks to a tolerable level. In other words, the objective is to lower $r^{\text{u}}_{t_i}$ of threat t_i beneath the tolerable risk $r^{\text{tol}}_{Z_i}$ by applying countermeasures. These measures result in security requirements for which the SUC operator must integrate concrete technical solutions. Only then can the SUC be considered secure with reference to the threats from ZCR 5.1.

ZCR 5.8 until ZCR 5.12 can be executed in a loop in which countermeasures are identified and then applied to the SUC. After doing so, each threat's likelihood and impact are reevaluated in ZCR 5.9, resulting in a residual risk (ZCR 5.10). This risk is compared with r^{tol} in ZCR 5.11, and additional countermeasures are identified in ZCR 5.12 if the residual risk is still not acceptable. The detailed risk analysis ends in ZCR 5.13, where the results are documented and made available.

The ISA-62443-3-2 specifies that requirements may encompass both technical and non-technical aspects, such as policies and procedures. Our focus, however, is specifically on technical requirements. In the course of this work, we will develop solutions tailored to these technical requirements and partly integrate them into the SUC.

The ISA-62443-3-3 furnishes an extensive overview of countermeasures for each foundational requirement, contingent upon the security level. Nonetheless, many of these countermeasures may not be directly applicable to our SUC because the ISA-62443 series was conceived for IACSs rather than automotive use cases. For example, a security level SL-2 for the IAC requirement necessitates a Public Key Infrastructure (PKI), a challenging demand for in-vehicle platforms comprising resource-constrained embedded systems. The complexity of maintaining long chains of public key certificates, especially on resource-constrained embedded systems, renders such an approach less practical. Therefore, we sought alternative solutions that are lighter and more resource-saving.

A first step towards alternative solutions was the work by El-Rewini et al. [70], which extensively surveyed automotive threats and countermeasures. In addition, the UNECE R 155 [63] is particularly helpful as it officially presents numerous mitigation techniques in part B of the fifth annex. Yet, we initially use ISA-62443-3-3 to identify countermeasures depending on the target security level SL-T_{Z_i} and only in the aftermath consult the UNECE R 155 in case a specific technical requirement is not applicable to our SUC. In the following, we give an overview of these countermeasures that are the security requirements for our SUC:

System Integrity One of the most critical security requirements for our SUC is system integrity. As a result of our threat analysis, the SL-T_{Z_i} indicate that all zones and conduits require the highest possible integrity protection. In ISA-62443-3-3, system integrity is a broad concept encompassing communication, software, and control units covering the foundational requirements of SI and IAS. More precisely, we identify the following security requirements:

1. **Software Integrity:** It must be determinable for any software whether it corresponds to the original or has been manipulated. This is especially crucial for updatable software components entering the vehicle from external sources. Violations of software integrity must be logged and reported.

2. **Communication Integrity:** Data transmission must be protected against network attacks aiming at manipulation, eavesdropping, or the insertion of false data. This protection must be maintained even during physical contact between the attacker and the bus. In addition to data integrity, its verifiable authenticity must be ensured so the message recipient can be confident of receiving data from a trusted source.

3. **Boot Process:** The integrity of the boot process of ECUs must be guaranteed to prevent the loading of malicious software during startup. This requirement applies especially to HPC platforms like the brainstem, cerebrum, or sensor modules of our SUC, which are equipped with a full-stack operating system. A technical solution is secure boot, which the boot loader must support.

4. **Malware Checks:** Any software must pass a malware check before its deployment into the SUC. This significantly reduces the risk of infected software components entering one or more vehicles. The verification must occur offline, and updates must also be considered.

5. **Physical Tamper Resistance:** ECUs must be protected against physical manipulation such as unauthorized replacement. Furthermore, they must have mechanisms for the secure storage of cryptographic tokens used, such as identity verification or secure communication. Hardware Security Modules (HSMs), which are also integrated into the AUTOSAR standard, maybe a technical solution

In addition to these requirements, ISA-62443-3-3 prescribes additional ones, particularly at a high-security level of SL-3 or SL-4. However, we find many of these requirements unnecessary or simply not applicable to our SUC. Notable is the validation of user inputs, which exists in the SUC only to a minimal extent in the HMI.

Data Confidentiality Protecting against eavesdropping and the leakage of sensitive data is another crucial security requirement for the SUC. In particular, transmitting sensor, environmental, and positional data can cause privacy issues as they reveal insights into the passenger's identity and behavior. Additionally, potentially confidential data such as bank information or login credentials may be stored on the HMI and must be protected against unauthorized access.

Specifically, data confidentiality in our SUC is required in zones Z_1, Z_2, and Z_3, as indicated by the foundational requirement *DC*. These zones encompass the ECUs along the main event chain, such as the brainstem, cerebrum, sensor, and dynamics modules. The requirement for data confidentiality applies to the latter because even rudimentary control parameters such as steering angle and torque can be used to reconstruct a vehicle's trajectory [71]. We derive the following security requirements for the SUC:

1. **Encryption**: Data transmission between applications on different ECUs must be encryptable to prevent unauthorized access to sensitive material such as environmental and positional data. To maximize efficiency, encryption should be selectively employed.

2. **Hardware Acceleration**: To minimize latency and achieve optimal performance, symmetric cryptography should be used, and hardware accelerators should provide corresponding primitives.

3. **Data Persistence**: Sensitive data must be completely and irreversibly deleted after processing on ECUs.

Identification Our threat analysis highlights the importance of protecting against unauthorized access to ECUs, which is necessary to avoid identity theft, the deployment of malicious software, and data leakage. This necessity is expressed by the foundational requirements *IAC* and *UC* in all zones, resulting in the following security requirements:

1. **User Identification**: Access to ECUs must only be granted to authorized entities. Non-embedded systems, especially, must have a password-protected user management system to prevent unauthorized access.

2. **Software Process**: Software components need the ability to identify each other. For this purpose, an identity per component is necessary and must be assigned no later than during integration. Unknown software, for example, uploaded by the user to the infotainment system, must also have an identity.

3. **Access Control**: Authorized entities must possess explicit permissions defining what actions are permitted, a requirement. Access control is particularly relevant for regulating access to sensitive sensor data in Zone Z_1, determining the rights of identified users using the HMI in Zone Z_5, and controlling authorized external access through the control room in Zone Z_6.

Especially with the IAC requirement, it becomes apparent that the ISA-62443 standard series was not explicitly designed for automotive systems but for IACS, where numerous human individuals have access to control systems, and portable devices are also regularly connected and disconnected to the network. For this reason, the requirements of ISA-62443-3-3 are tailored to scenarios in which a person gains access to a system component, which is only necessary in the maintenance case for our SUC. Therefore, we refrain from requirements such as multifactor authentication for all networks or a PKI within the vehicle.

Segmentation For zone Z_1 and conduit C_1, segmentation is a crucial security requirement to separate safety-critical communication from non-critical traffic and prioritize them concurrently. On HPC platforms, applications must also be segmented to prevent negative mutual influence, especially in case of corruption. The following security requirements have been identified:

1. **Network Segmentation**: Physically or logically segmenting the SUC networks can lower the number of entry points and significantly reduce the attack surface. Attacks such as privilege escalation or lateral movements can be prevented. Simultaneously, it simplifies monitoring, prioritization, and access control of data, ultimately leading to higher network efficiency. Our SUC consists of a central Ethernet network in C_1 organized in a ring topology, including both highly critical traffic like control parameters from the cerebrum to the dynamic modules and non-critical user inputs from the HMI. Critical traffic must be isolated and prioritized. This makes it more difficult for attacks to propagate through the vehicle. Since a physical separation is not feasible due to the need for a compact wiring harness, network segmentation must occur on a logical level, for example, through VLAN. Therefore, switches and the vehicle gateway must support configurability for this purpose.

2. **Application Partitioning**: Similar to network traffic, applications on ECUs should be isolated from each other. This prevents potential security breaches from affecting other applications and allows for more efficient resource utilization. Virtualizing applications is a suitable technical solution for this purpose. A unified environment enables better coexistence of legacy applications, more efficient development and testing, and improved responsiveness to regulatory requirements. The SUC has different HPC platforms in zone Z_1 and Z_4 supporting the execution of multiple applications

3. **Zone Boundary Checks**: Access control and monitoring must be possible at zone boundaries. For this purpose, ISA-62443-3-3 calls for managed interfaces such as switches, routers, and dedicated firewalls. Regarding our SUC, the transition from conduit C_1 to conduit C_4 must be controllable, as this is the vehicle's interface to the outside world. The vehicle gateway must, therefore, support configurable control flow monitoring and a robust logging mechanism.

DoS Protection Threats that attempt to restrict the availability of the SUC conduits have been identified for all conduits of the SUC. This typically occurs through DoS attacks, where a network is flooded with specific messages, causing a significant increase in latency. Especially for time-critical use cases, high latency is unacceptable, leading to the derivation of the following security requirements for the SUC.

1. **Rate Limiting**: Mitigating the consequences of a DoS attack can be achieved by limiting the transmitted data. However, it is crucial not to violate critical time guarantees

for transmitting critical traffic. Such limitation must, therefore, be finely adjustable and possibly applicable only to specific network areas.

2. **Firewall**: The central gateway, through which external connections enter the vehicle, must have a firewall that filters or even wholly blocks traffic. A strict deny-by-default policy for access control is necessary.

Monitoring The ability to detect and react to security incidents is necessary within all zones, expressed through the foundational requirement *TRE*. In particular, we derive the following security requirements for our SUC:

1. **Continuous Monitoring**: The SUC must be capable of detecting security-critical incidents. Such detection mechanisms should not only report failures during the verification of cryptographic protection but also identify network anomalies to detect DoS attacks in the SUC.

2. **Logging and Reaction**: Security incidents must be logged and made available to authorized entities. That means ECUs must provide an audit log locally and to an authorized third party. To protect the passenger's safety, we extend the requirement to make the SUC capable of adequately responding to a security incident, such as stopping or slowing down the vehicle.

With these security requirements protecting the SUC, we reevaluated the likelihood and impact of the threats in ZCR 5.9, resulting in a tolerable risk for each threat (ZCR 5.10 and ZCR 5.11). The security requirements must be technically realized while considering potentially conflicting requirements from other areas, such as the safety domain.

4.4 Discussion

Our threat and risk analysis particularly underscores the need for system and communication integrity, authentication, network and application segmentation, and the SUC's ability to respond to security incidents. Software and communication integrity, in particular, play crucial roles in achieving a secure automotive system. This finding aligns with previous research [68], which identifies the injection of fake messages as one of the most severe threats to modern vehicles. In the following discussion, we elaborate on various aspects that became evident during our risk assessment and the derivation of security requirements:

4.4.1 Applicability of ISA-62443

No established automotive standard existed when we started the threat and risk assessment. Hence, we opted for the ISA-62443 standards, even though they were initially designed for IACS rather than a software-defined vehicle. ISA-62443-3-2 provides a comprehensive risk analysis scheme for systematically identifying and documenting security requirements. However, certain aspects, such as the computation of security levels, are open to interpretation. ISA-62443-3-3 presents security requirements based on pre-determined security levels tailored for systems with regular interactions between humans and computing units. While these requirements are generally relevant, some, like two-factor authentication and a PKI for in-vehicle systems, may be unnecessary or inappropriate for the automotive domain. Thus, a reevaluation of the compensating measures given in ISA-62443-3-3 is essential for automotive system.s Today, UN regulation No. 155 [63] presents measures specifically designed for application in the automotive field.

The risk analysis in ISA-62443-3-2 relies on identifying and assessing threats, with each threat assigned a risk value. Compensating measures are then iteratively applied to reduce these risks to a tolerable level. However, the standard does not explicitly guide the determination of security levels. A future standard should clarify the relationship between risk and security levels to establish a common understanding.

Lastly, the ISA-62443 standards approach the SUC purely from a security perspective. Consequently, some security requirements must be evaluated, considering potential conflicts with requirements from other domains. For instance, the need to encrypt low-level driving commands in zone Z_2 addresses privacy concerns but introduces additional latency, a critical factor for safety. Ultimately, it will be a matter of weighing up which goal is more important and which measures should be implemented.

If adopting the risk analysis of ISA-62443-3-2 for the automotive domain, we suggest the following adaptations:

1. **Common Evaluation Criteria:** Establish a standard set of evaluation criteria and a consistent scoring scheme for automotive systems to enhance the comparability of analysis results. Consider prioritizing assessments using a multi-criteria decision-making process, such as AHP.

2. **Tailored Countermeasures**: Adjust the countermeasures listed in ISA-62443-3-3 for the automotive domain. Focus on lightweight and resource-saving techniques instead of user-oriented, computationally heavy systems. A good starting point is the recently published UN Regulation No. 155.

3. **Security-Safety Co-Engineering:** Extend foundational requirements to include safety requirements such as reliability, timing constraints, and redundancy for a security-safety co-engineering approach. This integration can address hazardous situations with outcomes similar to certain cybersecurity threats in a unified process.

In conclusion, we find the risk analysis of ISA-62443-3-2 applicable to automotive systems with the considerations outlined above. The partition into zones and conduits effectively breaks down the system's complexity for a more manageable analysis. Additionally, the concept of conduits enables modeling the propagating effects of attacks through the in-vehicle network.

4.4.2 Quality of Assessments

To obtain reasonable and consistent assessments of the SUC, we engaged an expert committee of eight computer scientists and mathematicians affiliated with the Security Engineering Group at the Technical University of Darmstadt.

The committee conducted thorough discussions for each assessment task, covering various aspects such as the relevance of evaluation criteria in the AHP processes, identifying threats, and determining security target levels. To commence this process, our experts underwent a detailed introduction to the SUC through a Q&A session.

The expert committee jointly accomplished the threat identification process and all assessments. We argue such a consensus-based approach effectively addresses subjectivity and vagueness, ensuring a well-rounded and reliable evaluation. It is worth noting that increasing the committee's heterogeneity, particularly in terms of educational background, might have further enhanced the robustness of our results.

4.4.3 Comparison to ISO/SAE 21434

Today, the ISO/SAE 21434 standard provides a risk analysis process for automotive systems. While the detailed risk analysis of ISA-62443-3-2 begins with identifying threats, the ISO/SAE 21434 starts from potential damage scenarios and traces them back to attack paths. More precisely, the risk assessment methods comprise seven steps (I-VII). Initially, damage scenarios that may occur through compromised assets are identified (I). A damage scenario is triggered by a set of adverse actions, a so-called threat scenario, enumerated in (II). The impact of each damage scenario is assessed according to four core categories of consequences: *Safety*, *Financial*, *Operational*, and *Privacy* (III). Subsequently, each threat scenario is decomposed into attack paths in a top-down or bottom-up approach (IV). The feasibility of each path is assessed according to a pre-defined scale (V), resulting in a risk value (VI) for each threat scenario, which also incorporates the impact of the damage scenario. Finally, risk reduction methods shall be realized (VII). If the risk for a threat scenario has to be reduced, a Cybersecurity Assurance Level (CAL) reveals requirements for the affected item.

At first glance, the risk analysis process of ISO/SAE 21434 and ISA-62443 have little in common. On closer inspection, however, both standards do share similar concepts. The CAL is similar to the SL-T values, which are only determined if the risk is too large. Instead of conduits, attack paths cover the propagating effects of adverse actions. While the idea of decomposing threat scenarios into attack paths is the most prominent feature of ISO/SAE 21434, our work reveals requirements that still need to be met by ISO/SAE 21434. Unlike ISA-62443, the novel automotive standard suggests assessment criteria and parameters in its annex. However, it insists on neither underlying cybersecurity requirements nor mitigation techniques, contrary to ISA-62443. Suggestions of countermeasures for specific CALs are helpful for a common minimum security perception because road vehicles are generally subjected to the same safety and legal requirements. Also, consistent scoring schemes and a dedicated process to identify relevant critical assets of a potentially complex architecture would be desirable.

4.5 Sub-conclusion

In this chapter, we presented a consensus-based implementation of the generic ISA-62443 cybersecurity standard for a software-defined vehicle. Our work involved the identification of risk evaluation criteria and the application of a multi-criteria decision-making scheme to assess automotive risks comprehensively. Additionally, we introduced a hierarchical threat model to facilitate a collaborative threat identification process, enabling experts to identify threats at a granular level for better comparability.

Based on the safety aspects and high-level security risks, we divided the SUC into zones and conduits and determined security levels for each as a vector, with each element representing the extent to which a foundational requirement is affected in the corresponding zone. Leveraging ISA-62443-3-3, we derived security requirements for the zones and conduits of the SDV, effectively mitigating risks to a tolerable level. Despite ISA-62443 initially targeting IACS, we advocate for its applicability to the automotive domain, especially when combined with the adaptations proposed in this work.

Addressing our first research question (RQ1), we conclude that system integrity, authentication, access control, the separation and prioritization of critical and uncritical networks and applications, and the ability to respond to security incidents are crucial for a secure SDV. Recognizing

that security is not free and introduces latency and maintenance costs is essential. Therefore, adopting a security-safety co-engineering approach is vital, considering requirements from different domains.

Concerning maintenance, SDVs not only require effective countermeasures against cyberattacks but also need integration into an overarching security process. This ensures the ability to update and maintain the system cost-efficiently, even after years of operation. In the subsequent chapters, we will present technical solutions for selected security requirements and integrate them into the UNICARagil vehicles.

5. Securing Signal-Based Protocols

The security requirement analysis conducted in Chapter 4 revealed that system integrity and authenticity are crucial prerequisites for a secure and safe SDV. This finding aligns with the cyberattacks presented in Section 3.1, which exploit software vulnerabilities to gain access to a poorly secured communication bus, eventually allowing the attacker to inject manipulated driving commands. Therefore, a crucial step in securing SDVs is to ensure that attackers can neither manipulate nor inject messages to in-vehicle networks, particularly if they transport safety-critical commands.

This chapter provides technical solutions to ensure authentic communication using signal-based protocols. Specifically, we focus on the Controler Area Network (CAN) and FlexRay, two well-established automotive communication protocols. While SDVs usually rely on Ethernet networks to transport large amounts of data, legacy protocols will not disappear immediately but rather be slowly replaced during the transition from classical vehicles to SDVs. This is why holistic security engineering must also consider them and investigate means to protect traffic from unauthorized access. Both CAN and FlexRay are deployed in isolated places in our reference vehicle.

The CAN protocol is one of the most famous signal-based legacy protocols. Researchers have demonstrated its susceptibility to cyberattacks in the last decade since vehicles are equipped with an increasing number of interfaces and undergo a technological transition, as elaborated in Chapter 1. Note that the CAN protocol has been deployed to road vehicles since the late 1980s. Due to its popularity and wide application area, many security solutions have been presented, aiming to authenticate CAN traffic efficiently. However, the protection against manipulation is only one side of the coin. An adequate key management process is necessary for a modular and dynamic software environment, as envisioned for SDVs. Therefore, this chapter presents a key computation and distribution scheme for CAN networks.

Regarding FlexRay, we elaborate on how protocol intrinsic properties can be used to authenticate traffic. Typically, FlexRay is used for highly critical and deterministic communication, such as x-by-wire applications. In addition, we investigate how cryptographic keys can be efficiently and securely updated, especially in a long-living vehicular system. Our reference vehicle deploys FlexRay as a fallback network for the four dynamic modules that actuate the wheels.

Please refer to Section 2.2 and Section 2.3 for more information on CAN and FlexRay. The further course of this chapter is structured as follows: Section 5.1 defines the attacker model, and Section 5.2 describes the system model. Subsequently, we present our key computation and distribution scheme for the CAN protocol in Section 5.3, followed by protection means for FlexRay networks in Section 5.4.

5.1 Attacker Model

Our works on securing CAN and FlexRay communication assume that the attacker $\mathcal{A}dv$ behaves like a typical Dolev-Yao attacker [72]. This attacker has significant network control and can monitor, publish, delete, and modify traffic on both CAN and FlexRay buses. The methods through which $\mathcal{A}dv$ gains control include physically connecting to the bus or exploiting software vulnerabilities on ECUs. Installing a fake ECU in a dishonest repair shop or connecting to an unsecured diagnostic interface are examples of the first case. The second case encompasses known vulnerabilities a (remote) attacker could exploit to access the bus through an honest ECU.

In CAN networks, $\mathcal{A}dv$ can communicate once access to the bus is established. This implies that $\mathcal{A}dv$ can potentially introduce malicious messages, modify existing ones, or disrupt the normal flow of communication within the CAN network.

Manipulating FlexRay traffic is more challenging, as $\mathcal{A}dv$ first needs to learn the network parameters required to participate in communication without violating the communication schedule. This complexity stems from the deterministic nature of FlexRay, where communication occurs according to a predefined schedule. Therefore, $\mathcal{A}dv$ faces a higher barrier to entry in interfering with FlexRay traffic compared to CAN.

Despite the significant control $\mathcal{A}dv$ has over the network, explicit limitations exist. $\mathcal{A}dv$ cannot break cryptographic primitives, suggesting that the security of cryptographic mechanisms is assumed to be intact. Additionally, $\mathcal{A}dv$ cannot actively corrupt ECUs, such as reading secrets from secure memory or installing software.

5.2 System Model

Our system model assumes signal-based protocols within small subnetworks, serving a specific use case and supporting legacy technology. In our reference vehicle, the CAN bus connects nearfield radar sensors in a line topology, while the FlexRay network acts as a fallback communication method for the highly safety-critical dynamic modules responsible for the wheels.

Traditionally, embedded ECUs in legacy automotive networks were designed for a specific task without clear hardware and software separation. Therefore, we assume that ECUs communicating through CAN or FlexRay are rather resource-constrained and highly integrated while still allowing for a more resourceful ECU to manage security measures.

We further assume the existence of a trusted security platform, $\mathcal{E}_{\mathrm{C}}$, that any ECU can communicate with. $\mathcal{E}_{\mathrm{C}}$ can be a dedicated ECU or a specially secured binary running on an HPC platform. Conceptually, $\mathcal{E}_{\mathrm{C}}$ does not need to be inside the vehicle if direct communication is feasible with the ECUs. However, we suggest integrating $\mathcal{E}_{\mathrm{C}}$ inside the vehicle for practical and scalability reasons.

The term "trusted" implies that $\mathcal{E}_{\mathrm{C}}$ functions as a root of trust, meaning that $\mathcal{A}dv$ neither compromises nor controls $\mathcal{E}_{\mathrm{C}}$. $\mathcal{E}_{\mathrm{C}}$ stores cryptographic secrets of ECUs that are part of the CAN and FlexRay network and are available whenever requested. To achieve this, $\mathcal{E}_{\mathrm{C}}$ must be equipped with fault-tolerant mechanisms to operate even during failures and robust protection against attacks. Though the task of designing and securing $\mathcal{E}_{\mathrm{C}}$ is orthogonal to our initial motivation of creating a secure key distribution protocol and ensuring traffic authenticity, we provide some insights: $\mathcal{E}_{\mathrm{C}}$ should be redundantly deployed to reduce the risk of failures and physically secure

through HSMs offering secure storage for cryptographic secrets. Note that hardware security and HSMs, in particular, belong to the requirements we identified in Section 4.3.3. The trust can be distributed across multiple instances of $\mathcal{E}_C$ if necessary.

In our case, the brainstem is a promising candidate for $\mathcal{E}_C$ due to its redundant layout and fail-operational position within the vehicle. Additionally, its ARM processors, equipped with a TrustZone, offer a secure environment for executing trusted code.

5.3 A Key Distribution Scheme for CAN Networks

The following sections are based on the publication "Using Implicit Certification to Efficiently Establish Authenticated Group Keys for In-Vehicle Networks" [2], which was presented at the Vehicular Networking Conference in 2019.

This section describes a modular key establishment scheme for in-vehicle networks using implicit certificates for lightweight key authentication. We derive *authenticated* elliptic-curve key pairs from previously distributed implicit certificates for each ECU during initialization. Next, we compute symmetric keys for those ECUs subscribed to the same CAN message identifier. As stated in [44], such a key strategy is the most promising one regarding scalability. We demonstrate how an One-Time Pad (OTP) can efficiently transmit keys and what precautions are necessary to keep the protocol secure. We consider our scheme as a preceding step to the numerous frameworks [73, 41, 42, 43, 44] that have been presented in recent years to authenticate and encrypt in-vehicle traffic. Since many existing solutions lack a key distribution scheme, we address the necessity for such a scheme.

Digital certificates are a common way of proving ownership of a cryptographic key to another digital entity. Typically, their application requires a PKI, resulting in a chain of trust represented by a series of dependent certificates. Consequently, each ECU would need to store a potentially long chain of certificates, which contradicts not only our system model that assumes resource-constrained ECUs and strives for a high degree of maintenance. Therefore, our key distribution scheme uses implicit certificates as a lightweight computational version. That way, assigning a digital identity to resource-constrained ECUs facilitates deployment in an SDV.

5.3.1 Implicit Certificates

A certificate binds an identity to a cryptographic key, thus allowing it to prove key authenticity. Usually, this is done by a trusted Certificate Authority (CA) that signs the identifier and the key. If a third party trusts this CA, it will also trust the ownership of the key. Explicit certificates contain information about the owner, the key, and a digital signature by the CA. They are a well-established tool on contemporary computer systems. In most cases, explicit certificates are available in the X.509 format. Since a third party typically considers only a small number of CAs as trustworthy, a chain of trust is built to verify a certificate. Such a chain of trust relies on transitive trust, i.e., if a party A trusts CA and CA certifies CA' then A trusts CA' as well. Verifying chains of explicit certificates imposes a comparably large computational and memory overhead, making them inconvenient for embedded in-vehicle components.

In comparison, an implicit certificate is a lightweight cryptographic primitive that also allows one to verify the identity behind a key. It enables the certificate owner to derive the public key by using the public key of the trusted authority. The certificate owner can exclusively compute the corresponding secret key. Evidence of key authenticity is implicitly guaranteed

once the certificate owner uses the private key corresponding to the derived public key. Implicit certificates are also called micro certificates because they are generally smaller and faster than explicit ones. Assuming a key length of 128 bits, an implicit certificate is 23 times smaller than the explicit equivalent [74]. For this reason, we investigate in this work to what extent they apply to in-vehicle networks to authenticate shared keys between ECUs.

5.3.2 Notation

In the subsequent sections, we use the following notation: Let ECU be the set of ECUs with $\mathcal{E}_\mathrm{i} \in \mathsf{ECU}$ representing a specific ECU participating in our scheme. As mentioned, $\mathcal{E}_\mathrm{C}$ is a trusted component functioning as a CA. The set $\mathcal{C}$ encompasses all CAN message identifiers with $\mathcal{C}_{\mathcal{E}_\mathrm{i}}$ indicating the message types to which $\mathcal{E}_\mathrm{i}$ is subscribed. $\mathcal{E}^{\text{CAN-ID}}$ refers to the set of ECUs subscribed to CAN messages of type CAN-ID $\in \mathcal{C}$. The elliptic-curve private key of $\mathcal{E}_\mathrm{i}$ is represented by $d_{\mathcal{E}_\mathrm{i}}$, while $Q_{\mathcal{E}_\mathrm{i}}$ is the corresponding public key.

5.3.3 Assumptions

We assume a static network topology, as described in our system model in Section 5.2. It is expected that each ECU possesses SRAM memory. This assumption is essential, as we leverage the physical properties of SRAM memory to retrieve a unique key using a Physical Unclonable Function (PUF). Given that legacy ECUs typically have limited resources and are highly integrated systems, we avoid the requirement for secure memory. However, we do assume that all ECUs participating in our scheme possess the required cryptographic metadata, including a common generator G for the elliptic curve P-256, a secure hash function *Hash*, and a Key Derivation Function (KDF) denoted as *KDF*, where *KDF*(k,i) refers to the derivation of a key through i iterations from the seed sd. Depending on the chosen *Hash*, the *KDF* can be constructed iteratively from *Hash*. Finally, it is important to note that no initial level of trust is preassumed between the ECUs.

5.3.4 Scheme Phases

Our key distribution scheme comprises three main phases that must be executed sequentially. Since we do not expect an initial level of trust between the ECUs, we first mutually authenticate each $\mathcal{E}_\mathrm{i} \in \mathsf{ECU}$ with $\mathcal{E}_\mathrm{C}$ using a PUF. In the second phase, each legitimate ECU $\mathcal{E}_\mathrm{i}$ receives an implicit certificate $IC_{\mathcal{E}_\mathrm{i}}$, which allows to prove authenticity to others ECUs. This step is crucial to prevent man-in-the-middle attacks in subsequent steps. The third phase involves computing a symmetric key for each CAN message type using Elliptic Curve Diffie-Hellman (ECDH). Note that after performing the first two phases, the third one can be repeated arbitrarily often, resulting in authentic message-based group keys. To summarize, the three phases of our scheme can be outlined as follows:

1. **ECU Authentication Phase**: Each $\mathcal{E}_\mathrm{i} \in \mathsf{ECU}$ contacts $\mathcal{E}_\mathrm{C}$, and they mutually authenticate using a PUF. By doing so, $\mathcal{E}_\mathrm{C}$ ensures to only issue implicit certificates for legitimate ECUs in the second phase. Similarly, each $\mathcal{E}_\mathrm{i}$ must ensure the authenticity of $Q_{\mathcal{E}_\mathrm{m}}$ to verify implicit certificates of other ECUs.

2. **ECU Certification Phase**: $\mathcal{E}_\mathrm{C}$ issues an implicit certificate $IC_{\mathcal{E}_\mathrm{i}}$ for each legitimate $\mathcal{E}_\mathrm{i}$. In the end, $\mathcal{E}_\mathrm{C}$ possesses an authenticated public key $Q_{\mathcal{E}_\mathrm{i}}$ of $\mathcal{E}_\mathrm{i}$.

3. **Key Computation Phase:** Finally, the $IC_{\mathcal{E}_\mathrm{i}}$ allows the ECUs from $\mathcal{E}^{\text{CAN-ID}}$ to compute a shared symmetric group key for messages of identifier CAN-ID. To achieve this, we establish an authentic session key using ECDH, which subsequently secures the transmission of further cryptographic keys.

Executing the first and second phases of the scheme is necessary only if a new implicit certificate is issued. This situation may arise after a predefined key lifetime, a firmware update, or replacing an ECU. Only the third phase must be rerun for a simple key update to secure CAN communication. However, all involved ECUs must be capable of storing the certificate $IC_{\mathcal{E}_\mathrm{i}}$ after shutdown. If they only possessed volatile memory, all three phases must be conducted whenever a key is required. We propose performing these phases for each new driving session, such as when the vehicle starts. In the following, we describe each scheme phase in more detail.

ECU Authentication Phase

This phase is designed to establish an initial level of trust between each $\mathcal{E}_\mathrm{i}$ and $\mathcal{E}_\mathrm{C}$. In subsequent steps, $\mathcal{E}_\mathrm{C}$ issues an implicit certificate only for those control units that have been successfully authenticated in this phase. The primary objective of this phase is to generate a shared symmetric key, denoted as k_{phy_i}, between $\mathcal{E}_\mathrm{i}$ and $\mathcal{E}_\mathrm{C}$, and to provide $\mathcal{E}_\mathrm{i}$ with $\mathcal{E}_\mathrm{C}$'s public key $Q_{\mathcal{E}_\mathrm{m}}$. $Q_{\mathcal{E}_\mathrm{m}}$ is necessary for verifying certificates of other ECUs. The key k_{phy_i} is derived by creating a physical fingerprint of $\mathcal{E}_\mathrm{i}$. Consequently, $\mathcal{E}_\mathrm{i}$ and $\mathcal{E}_\mathrm{C}$ only trust each other if both can demonstrate possession of k_{phy_i}.

To establish the initial level of trust, we use an SRAM PUF to generate and share k_{phy_i} based on the PUF response and a pre-agreed fuzzy extractor scheme. Since the PUF response resides on $\mathcal{E}_\mathrm{i}$, its challenge-response pair should be securely stored in $\mathcal{E}_\mathrm{C}$ at fabrication time. Since the unique SRAM PUF response is exclusively known to $\mathcal{E}_\mathrm{C}$, no other ECU can obtain k_{phy_i}. The benefit of using a PUF is that we do not need to deploy a cryptographic key on $\mathcal{E}_\mathrm{i}$ at fabrication time but instead solely rely on physical properties.

Figure illustrates 5.1 the ECU authentication phase. At vehicle startup, $\mathcal{E}_\mathrm{C}$ sends a randomly chosen nonce n, the challenge c and helper data hd to $\mathcal{E}_\mathrm{i}$. A challenge is the set of memory addresses that can be queried. Upon reception, $\mathcal{E}_\mathrm{i}$ queries its PUF with c as input and retrieves the PUF response. Using a fuzzy extractor, $\mathcal{E}_\mathrm{i}$ computes k_{phy_i} by XORing the PUF response with the helper data. Note that the fuzzy extractor removes noise from the PUF response. $\mathcal{E}_\mathrm{i}$ likewise selects a nonce l and sends $n + l$ encrypted with k_{phy_i} back to $\mathcal{E}_\mathrm{C}$. $\mathcal{E}_\mathrm{C}$ decrypts the received message, computes l, and sends its public key $Q_{\mathcal{E}_\mathrm{m}}$, l, and a unique identifier id_i encrypted with k_{phy_i} to $\mathcal{E}_\mathrm{i}$. id_i identifies $\mathcal{E}_\mathrm{i}$ and will be included in the implicit certificate. Since only $\mathcal{E}_\mathrm{C}$ can compute k_{phy_i}, $\mathcal{E}_\mathrm{i}$ trusts $Q_{\mathcal{E}_\mathrm{m}}$ after this step.

As indicated above, we deliberately do not use a pre-shared long-term key between $\mathcal{E}_\mathrm{C}$ and $\mathcal{E}_\mathrm{i}$ since we do not assume ECUs to be equipped with secure memory. However, ECU do have memory components that always have unique physical properties. At fabrication time, $\mathcal{E}_\mathrm{C}$ needs once to gain access to the PUF of $\mathcal{E}_\mathrm{i}$ to compute the corresponding helper data to k_{phy_i}, which is selected randomly.

ECU Certification Phase

In the second phase, as illustrated in Figure 5.2, $\mathcal{E}_\mathrm{C}$ issues an implicit certificate $IC_{\mathcal{E}_\mathrm{i}}$ to the previously authenticated $\mathcal{E}_\mathrm{i}$. Such micro certificates are not only smaller in size but require less

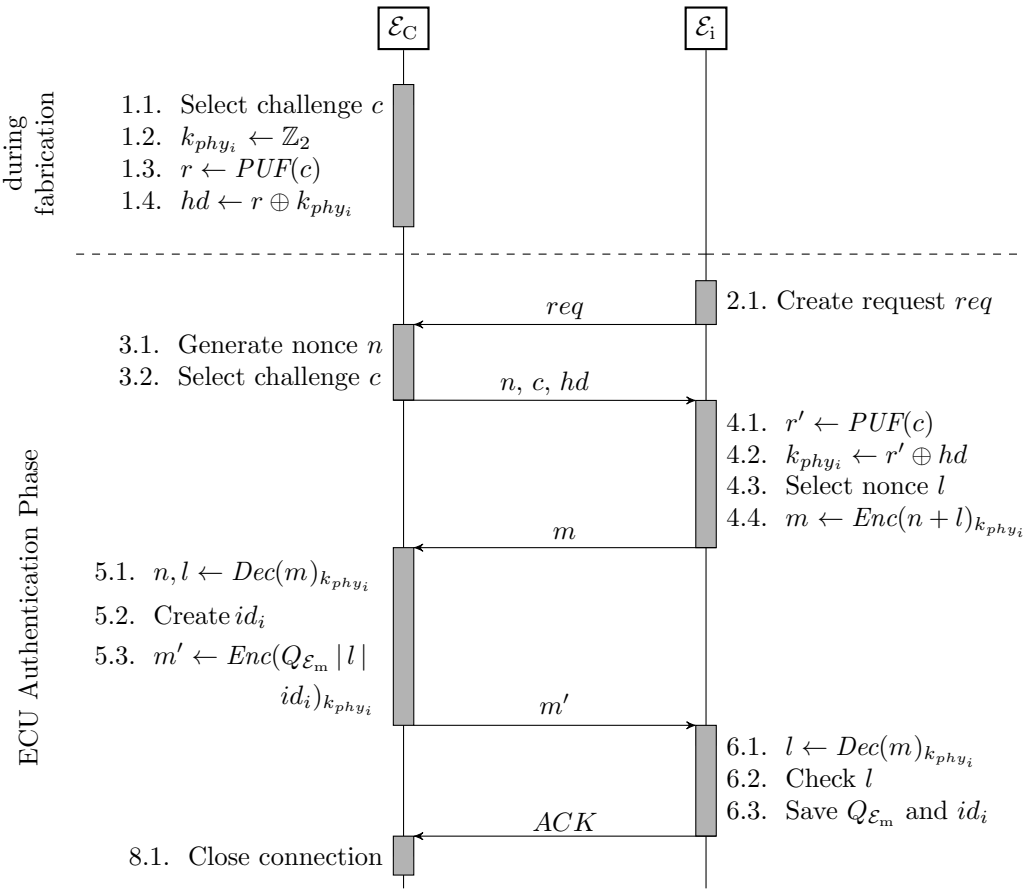

Figure 5.1: During the ECU Authentication Phase, $\mathcal{E}_\mathrm{C}$ and $\mathcal{E}_\mathrm{i}$ mutually authenticate. For this purpose, $\mathcal{E}_\mathrm{i}$ retrieves k_{phy_i} from its PUF by using a challenge c. $\mathcal{E}_\mathrm{C}$ sends its public key $Q_{\mathcal{E}_\mathrm{m}}$ encrypted to $\mathcal{E}_\mathrm{i}$. Since both $\mathcal{E}_\mathrm{i}$ and $\mathcal{E}_\mathrm{C}$ eventually possess k_{phy_i}, they trust each other.

computational resources compared to X.509 certificates. More precisely, $\mathcal{E}_\mathrm{C}$ distributes Elliptic Curve Qu-Vanstone (ECQV) certificates [75] to those ECUs authenticated in the first phase. An implicit certificate allows a third party to derive an authenticated public key $Q_{\mathcal{E}_\mathrm{i}}$ of $\mathcal{E}_\mathrm{i}$ using $Q_{\mathcal{E}_\mathrm{m}}$. Since the authenticity of $Q_{\mathcal{E}_\mathrm{m}}$ has been proven in the first phase, we do not need to verify potentially long and inefficient certificate chains. As mentioned earlier, this would be hard to achieve in an automotive system, including resource-constrained devices. Only after this phase $Q_{\mathcal{E}_\mathrm{i}}$ is used to compute CAN message-based group keys.

The ECU Certification Phase is triggered with an authentication request message sent by each $\mathcal{E}_\mathrm{i}$ that asks for an $IC_{\mathcal{E}_\mathrm{i}}$. This triggering message has a high-priority type, potentially overwriting other messages on the bus. The request contains the previously obtained identifier id_i. $\mathcal{E}_\mathrm{i}$ selects a random $x \in_R [1, o-1]$ and sends the product $x \cdot G$ to $\mathcal{E}_\mathrm{C}$, where G is a generator point on the elliptic curve P-256, and o is the order of G. $\mathcal{E}_\mathrm{C}$ creates the implicit certificate $IC_{\mathcal{E}_\mathrm{i}}$ for $\mathcal{E}_\mathrm{i}$. For this, it first calculates $w = x \cdot G + l \cdot G$, where $l \in_R [1, o-1]$, and then encodes w and id_i in $IC_{\mathcal{E}_\mathrm{i}}$. Additionally, it computes $h\,Hash(IC_{\mathcal{E}_\mathrm{i}})$ and $s = h \cdot l + d_{\mathcal{E}_\mathrm{C}}$. $\mathcal{E}_\mathrm{i}$ needs s to calculate its private key $d_{\mathcal{E}_\mathrm{i}}$ that corresponds to $IC_{\mathcal{E}_\mathrm{i}}$. $\mathcal{E}_\mathrm{C}$ encrypts s with k_{phy_i} and sends $IC_{\mathcal{E}_\mathrm{i}}$ and s back to $\mathcal{E}_\mathrm{i}$.

The encryption of s is not part of the original ECQV scheme. However, by decrypting s, $\mathcal{E}_\mathrm{i}$ implicitly proves its identity to $\mathcal{E}_\mathrm{C}$ because k_{phy_i} is only known to these two parties. Without this encryption step, the adversary $\mathcal{A}dv$ could easily impersonate $\mathcal{E}_\mathrm{i}$, leading to the erroneous issuance of a valid certificate. Any party can compututationally derive the public key $Q_{\mathcal{E}_\mathrm{i}}$ by calculating $Q_{\mathcal{E}_\mathrm{i}} = h' \cdot w + Q_{\mathcal{E}_\mathrm{m}} = h' \cdot (m + lG) + Q_{\mathcal{E}_\mathrm{m}} = h'xG + h'lG + Q_{\mathcal{E}_\mathrm{m}} = (h'x + hl + d_{\mathcal{E}_\mathrm{C}}) \cdot G = d_{\mathcal{E}_\mathrm{i}} \cdot G$. The

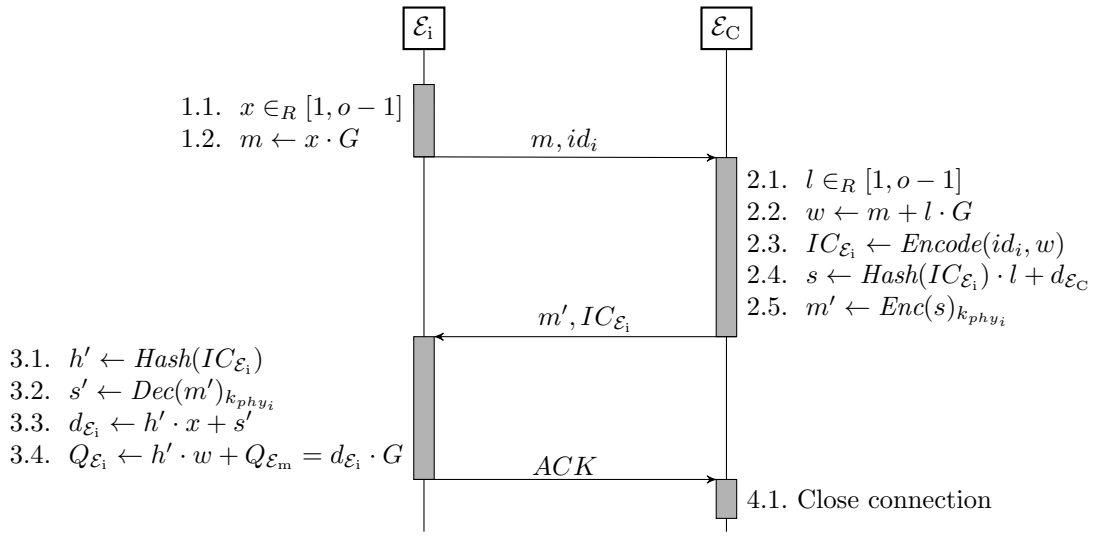

Figure 5.2: During the ECU Certification Phase, $\mathcal{E}_\mathrm{C}$ issues ECQV certificates to all previously authenticated control units $\mathcal{E}_\mathrm{i}$. Any ECU possessing $IC_{\mathcal{E}_\mathrm{i}}$ can derive the authenticated public key $Q_{\mathcal{E}_\mathrm{i}}$, while only $\mathcal{E}_\mathrm{i}$ can compute the corresponding private key $d_{\mathcal{E}_\mathrm{i}}$.

computation of the corresponding private key $d_{\mathcal{E}_\mathrm{i}}$ requires x, which is known only to $\mathcal{E}_\mathrm{i}$. More precisely, $\mathcal{E}_\mathrm{i}$ computes $d_{\mathcal{E}_\mathrm{i}}$ as $h' \cdot s'$, which is equivalent to $h' \cdot x + h \cdot l + d_{\mathcal{E}_\mathrm{C}}$ and therefore is the matching private key to $Q_{\mathcal{E}_\mathrm{m}}$.

After this phase, each legitimate $\mathcal{E}_\mathrm{i}$ possesses an implicit certificate $IC_{\mathcal{E}_\mathrm{i}}$ that allows any further party to derive an authenticated public key $Q_{\mathcal{E}_\mathrm{i}}$. In the third phase, our scheme uses $Q_{\mathcal{E}_\mathrm{i}}$ to create group keys for securing CAN messages. Unlike traditional explicit X.509 certificates, implicit certificates have no means of revocation. Therefore, we suggest defining a fixed expiration time, after which new certificates must be issued.

Key Computation Phase

In the final phase, we use the previously distributed implicit certificates to establish short-term symmetric keys for securing CAN frames of a specific identifier. In the end, a given $\mathcal{E}_\mathrm{i}$ receives $\left|\mathcal{C}_{\mathcal{E}_\mathrm{i}}\right|$ distinct cryptographic keys, with each key corresponding to a subscribed CAN message identifier. The role of $k_{\text{CAN-ID}}$ is to secure CAN frames of the type CAN-ID. Its primary functions include computing a MAC and, optionally, encrypting traffic. As CAN operates as a broadcast protocol without distinct recipients, $k_{\text{CAN-ID}}$ is made accessible to all ECUs in $\mathcal{E}^{\text{CAN-ID}}$. In other words, all ECUs communicating through the message identifier CAN-ID receive the same key $k_{\text{CAN-ID}}$.

Once the first two phases of the scheme are executed, the third phase can be rerun arbitrarily often, allowing for fast key updates with minimal overhead. However, for security reasons, we recommend executing all phases if the ECUs cannot securely store the private key $d_{\mathcal{E}_\mathrm{i}}$. Failing to do so may expose $d_{\mathcal{E}_\mathrm{i}}$ to theft, enabling an adversary to participate in the communication.

This phase consists of two steps: In step [A], we compute a session key $K_{\mathcal{E}_\mathrm{i}-\mathcal{E}_\mathrm{C}}$ between $\mathcal{E}_\mathrm{i}$ and $\mathcal{E}_\mathrm{C}$ using ECDH. Subsequently, in step [B], we utilize this key to securely transmit the keys to the ECUs subscribed to $\mathcal{C}_{\mathcal{E}_\mathrm{i}}$.

[A] $\mathcal{E}_\mathrm{i}$ sends a *session_key_request* to $\mathcal{E}_\mathrm{C}$ containing its implicit certificate $IC_{\mathcal{E}_\mathrm{i}}$. Upon reception, $\mathcal{E}_\mathrm{C}$ publishes a randomly chosen nonce $n' \in \mathbb{Z}_2^{128}$ and both $\mathcal{E}_\mathrm{i}$ and $\mathcal{E}_\mathrm{C}$ compute $K_{\mathcal{E}_\mathrm{i}-\mathcal{E}_\mathrm{C}} =$

$KDF(n' \cdot d_{\mathcal{E}_\mathrm{i}} \cdot Q_{\mathcal{E}_\mathrm{m}}) = KDF(n' \cdot d_{\mathcal{E}_\mathrm{C}} \cdot Q_{\mathcal{E}_\mathrm{i}})$. The key derivation function KDF serves as a high entropy source to obtain a uniformly distributed key string. Moreover, it hides potential structural information of the raw ECDH key.

[B] $\mathcal{E}_\mathrm{C}$ collects a random and secret $z_{\mathcal{E}_\mathrm{i}} \in \mathbb{Z}_2^{128}$ from each ECU $\mathcal{E}_\mathrm{i}$ after having broadcasted a session start notification. For that purpose, $\mathcal{E}_\mathrm{i}$ encrypts its $z_{\mathcal{E}_\mathrm{i}}$ using again the session key $K_{\mathcal{E}_\mathrm{i}-\mathcal{E}_\mathrm{C}}$ and sends the ciphertext to $\mathcal{E}_\mathrm{C}$. In response, $\mathcal{E}_\mathrm{C}$ decrypts $z_{\mathcal{E}_\mathrm{i}}$ and subsequently computes $k_{\text{CAN-ID}} = z_{\mathcal{E}_\mathrm{i}} \oplus z_{\mathcal{E}_{\mathrm{i}+1}} \oplus \ldots z_{\mathcal{E}_\mathrm{n}} \forall\, \mathcal{E}_\mathrm{i} \in \mathcal{E}^{\text{CAN-ID}},\ n = \left|\mathcal{E}^{\text{CAN-ID}}\right|$ for every CAN-ID $\in \mathcal{C}$. Finally, $\mathcal{E}_\mathrm{C}$ encrypts the keys and sends them back to $\mathcal{E}_\mathrm{i}$.

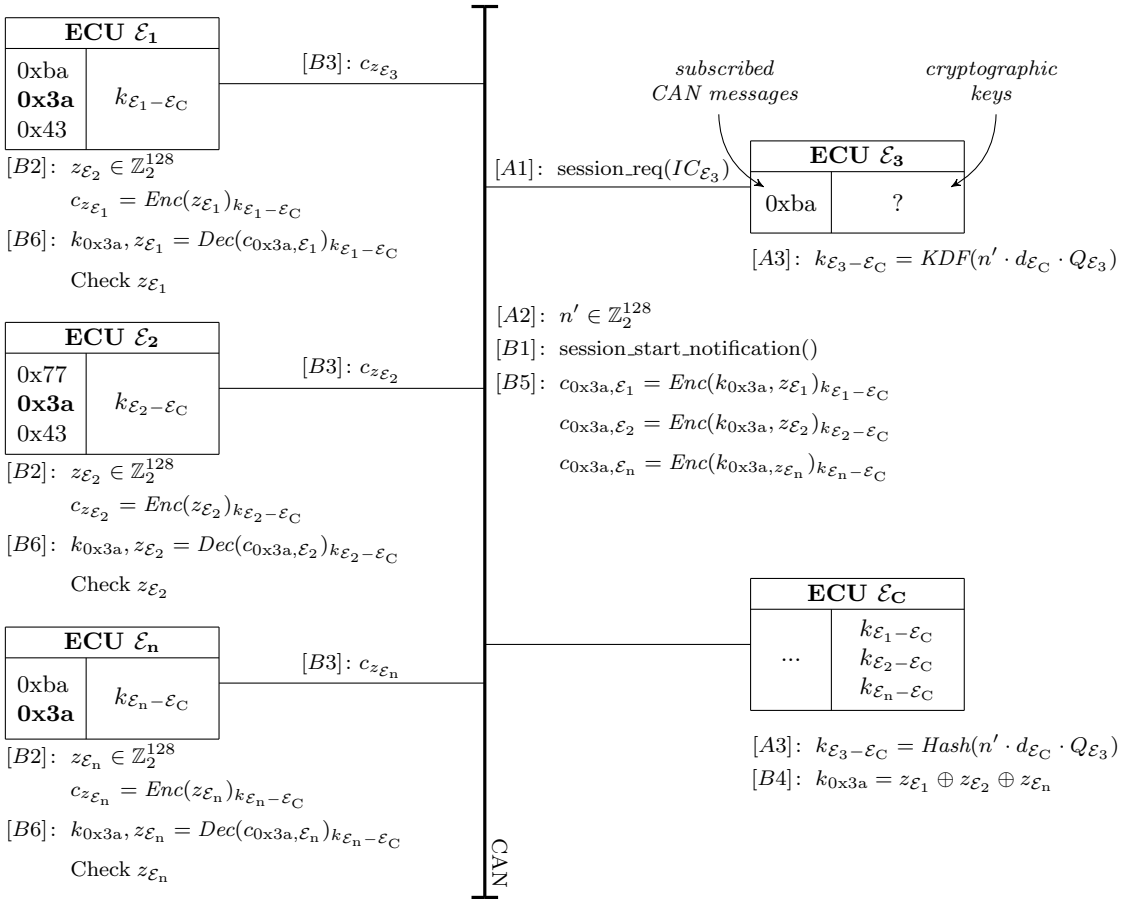

Figure 5.3: In stages [A1] to [A3] of step [A], $\mathcal{E}_3$ negotiates a session key $K_{\mathcal{E}_3-\mathcal{E}_\mathrm{C}}$ with $\mathcal{E}_\mathrm{C}$ using ECDH and its implicit certificate $IC_{\mathcal{E}_3}$. In stages [B1] to [B6] of step [B], $\mathcal{E}_\mathrm{C}$ initiates the creation of the symmetric key k_{0x3a} for the CAN-ID 0x3a. At first, $\mathcal{E}_1$, $\mathcal{E}_2$, and $\mathcal{E}_\mathrm{n}$ create each a secret $z_{\mathcal{E}_1}$, $z_{\mathcal{E}_2}$ and $z_{\mathcal{E}_\mathrm{n}}$, encrypt them with the session key [B2] and send the ciphertexts to $\mathcal{E}_\mathrm{C}$ [B3]. $\mathcal{E}_\mathrm{C}$ computes $k_{0x3a} = z_{\mathcal{E}_1} \oplus z_{\mathcal{E}_2} \oplus z_{\mathcal{E}_\mathrm{n}}$ [B4] and sends it encrypted to $\mathcal{E}_1$, $\mathcal{E}_2$, and $\mathcal{E}_\mathrm{n}$ [B5]. The ECUs decrypt k_{0x3a} [B6] and store it.

Figure 5.3 illustrates the individual steps to compute a $k_{\text{CAN-ID}}$. Note that each $k_{\text{CAN-ID}}$ is the XORed random numbers produced by the $\mathcal{E}_\mathrm{i}$ that are subscribed to CAN-ID. Consequently, those message types to which the same ECUs are subscribed receive the same key. An additional key would not significantly increase security, as the same parties communicate through the same bus.

Step [B] should be repeated every time the vehicle starts to get fresh keys. Once the implicit certificate is considered expired or the ECUs cannot securely store the keys, all phases must be rerun.

Using an OTP for Encryption and Decryption To protect the cryptographic keys associated with CAN message identifiers from eavesdroppers, we must transmit them encrypted from $\mathcal{E}_\mathrm{C}$ to the ECUs. We generically indicate this using the *Enc* command during the scheme's third phase.

Consequently, the involved ECUs must support an encryption and decryption primitive, preferably provided by a hardware crypto extension. The responsibility of determining the functionalities represented by *Enc* lies with the vehicle manufacturer. Depending on the chosen algorithm, slight modifications may be required in the scheme phase to ensure security.

One option is to adopt an OTP as a lightweight and simultaneously secure variant. When employed correctly, the OTP guarantees perfect secrecy, ensuring that the ciphertext does not disclose any information about the original message and remains indistinguishable from a random number. Another advantage is that implementation involves only an XOR operation, making it suitable for any microprocessor.

However, the key of an OTP must be used only once. Otherwise, its perfect secrecy gets lost, potentially leading to the revelation of the original message in the worst-case scenario. In our case, an OTP can only be correctly applied if a KDF is involved in step [B] of the third phase. This is necessary because ECU $\mathcal{E}_\mathrm{i}$ encrypts its random number $z_{\mathcal{E}_\mathrm{i}}$ using the shared key $k_{\mathcal{E}_\mathrm{i}-\mathcal{E}_\mathrm{C}}$. Later, $k_{\mathcal{E}_\mathrm{i}-\mathcal{E}_\mathrm{C}}$ is used to secure the transmission of the group key $k_{\text{CAN-ID}}$. Although each encryption itself is secure, the entire protocol would be broken if $k_{\mathcal{E}_\mathrm{i}-\mathcal{E}_\mathrm{C}}$ is indeed used twice, as we show in the following:

Consider a network comprising two ECUs, denoted as $\mathcal{E}_\mathrm{A}$ and $\mathcal{E}_\mathrm{B}$, engaged in communication on the message identifier $0xAB$. According to the proposed protocol, both ECUs independently choose random numbers, namely $z_{\mathcal{E}_\mathrm{A}}$ and $z_{\mathcal{E}_\mathrm{B}}$. $\mathcal{E}_\mathrm{A}$ computes $c_{z_{\mathcal{E}_\mathrm{A}}} = z_{\mathcal{E}_\mathrm{A}} \oplus k_{\mathcal{E}_\mathrm{A}-\mathcal{E}_\mathrm{C}}$ while $\mathcal{E}_\mathrm{B}$ similarly calculates $c_{z_{\mathcal{E}_\mathrm{B}}} = z_{\mathcal{E}_\mathrm{B}} \oplus k_{\mathcal{E}_\mathrm{B}-\mathcal{E}_\mathrm{C}}$.

Upon receiving these values, $\mathcal{E}_\mathrm{C}$ decrypts $z_{\mathcal{E}_\mathrm{A}}$ and $z_{\mathcal{E}_\mathrm{B}}$, computes $k_{0xAB} = z_{\mathcal{E}_\mathrm{A}} \oplus z_{\mathcal{E}_\mathrm{B}}$, calculates $c_{k_{0xAB},\mathcal{E}_\mathrm{A}} = k_{0xAB} \oplus k_{\mathcal{E}_\mathrm{A}-\mathcal{E}_\mathrm{C}}$ and $c_{k_{0xAB},\mathcal{E}_\mathrm{B}} = k_{0xAB} \oplus k_{\mathcal{E}_\mathrm{B}-\mathcal{E}_\mathrm{C}}$, and sends both ciphertext to the corresponding ECUs. Meanwhile, the attacker $\mathcal{A}dv$ intercepts $c_{z_{\mathcal{E}_\mathrm{A}}}$, $c_{z_{\mathcal{E}_\mathrm{B}}}$, $c_{k_{0xAB},\mathcal{E}_\mathrm{A}}$, and $c_{k_{0xAB},\mathcal{E}_\mathrm{B}}$. Recall that $\mathcal{A}dv$ possesses this ability according to the attacker model presented in Section 5.1. Subsequently, $\mathcal{A}dv$ can recompute the secret $z_{\mathcal{E}_\mathrm{B}}$ by XORing the ciphertexts:

$$\begin{aligned} c_{z_{\mathcal{E}_\mathrm{A}}} \oplus c_{k_{0xAB},\mathcal{E}_\mathrm{A}} &= z_{\mathcal{E}_\mathrm{A}} \oplus k_{\mathcal{E}_\mathrm{A}-\mathcal{E}_\mathrm{C}} \oplus k_{0xAB} \oplus k_{\mathcal{E}_\mathrm{A}-\mathcal{E}_\mathrm{C}} \\ &= z_{\mathcal{E}_\mathrm{A}} \oplus k_{0xAB}\mathcal{E}_\mathrm{C} \\ &= z_{\mathcal{E}_\mathrm{A}} \oplus z_{\mathcal{E}_\mathrm{A}} \oplus z_{\mathcal{E}_\mathrm{B}} \\ &= z_{\mathcal{E}_\mathrm{B}} \end{aligned}$$

Similarly, $\mathcal{A}dv$ could obtain $z_{\mathcal{E}_\mathrm{A}}$, eventually enabling $\mathcal{A}dv$ the recomputation of the key k_{0xAB}.

This example shows that using the same key more than once carries significant consequences if using an OTP for *Enc*. Therefore, rather than directly employing the key $k_{\mathcal{E}_\mathrm{A}-\mathcal{E}_\mathrm{C}}$, we recommend using it as a seed for a KDF. Besides, the KDF receives the number of iterations as input, indicating the count of internal hashing operations. This iteration count must vary each time step [B] is invoked unless a new implicit certificate is issued, leading to the generation of a fresh $k_{\mathcal{E}_\mathrm{A}-\mathcal{E}_\mathrm{C}}$.

5.3.5 Evaluation

In this section, we describe the implementation of the proposed scheme and then evaluate it using runtime measurements and a theoretical security analysis. We mainly focus on timing behavior since automotive networks typically must not have large latencies for safety reasons.

Central vs. Decentral

Our protocol relies on the presence of the central $\mathcal{E}_{\mathrm{C}}$, responsible for issuing certificates and computing cryptographic keys for CAN message types. Nevertheless, only minor adjustments to the proposed scheme are required to enable decentralized key computation. Once each ECU possesses an implicit certificate, it can establish secure connections with all others, facilitating the secure transmission of its secret z value. Eventually, all ECUs subscribed to a specific CAN message identifier can compute the corresponding key by XORing the received z values.

The deliberate choice of a central instance for key computation offers two primary advantages. Firstly, a central approach scales better regarding the number of transmitted messages. In a network with n ECUs, our central approach necessitates the establishment of n secure channels, whereas a fully decentralized approach would result in $\frac{n \cdot (n-1)}{2}$ connections. Since a typical legacy network in an SDV is not expected to be large, the scalability argument may not be convincing alone. Another rationale for centrally administering cryptographic keys is maintenance. In a decentralized scheme, each ECU requires system knowledge, necessitating awareness of other participating parties, which complicates system updates. In a central approach, ECUs remain system-agnostic and interact solely with $\mathcal{E}_{\mathrm{C}}$ without the need to engage with other ECUs. Consequently, changes to the security model only need to be implemented in $\mathcal{E}_{\mathrm{C}}$. From a maintenance perspective, such an approach is more promising as it is more cost-efficient.

Security Analysis

The presented scheme employs implicit certificates for distributing authenticated keys to ECUs, facilitating the rapid computation of shared symmetric keys. Initially, a session key is generated using ECDH. Implicit certificates and ECDH are based on the Elliptic Curve Discrete Logarithm Problem (ECDLP), considered hard [76]. While it is straightforward to compute a point $P = x \cdot G$ given x and G, determining x from P and G is computationally hard. As x remains confidential, $\mathcal{A}dv$ can neither forge implicit certificates nor spoof a session key.

We suggest using OTPs to compute group keys for CAN message types for efficiency reasons. Assuming a key length of 128 bits makes brute-force attacks difficult, requiring 2^{128} attempts. The integrity of our scheme relies on keeping k_{phy} secure. This key is never disclosed and is exclusively stored on $\mathcal{E}_{\mathrm{C}}$, with each $\mathcal{E}_{\mathrm{i}}$ deriving it from unique physical properties.

The authenticity of keys is based upon the implicit certificates, while nonces ensure freshness, thereby enabling the detection of replay attacks.

Implementation

We connected seven exemplary ECUs in both a classical CAN network and a CAN-FD network. As detailed in Section 2.2, CAN-FD offers a higher bandwidth of up to 8 Mbit/s compared to classical CAN (1 Mbit/s). A CAN-FD frame supports a payload of 64 bytes, while a classical CAN frame is limited to 8 bytes.

For the classical CAN network, we utilized Olimex ESP32-EVB boards as an evaluation platform. These boards feature 4 MB flash memory, a 240 MHz 32-bit microprocessor, and an integrated CAN interface. In the case of the CAN-FD network, Raspberry Pis (RPis) 3B+ with a PiCAN-FD shield were employed. These RPi boards have 1 GB RAM and a 1.4 GHz ARM Cortex-A53. While acknowledging that RPi boards cannot be classified as low-resource ECUs, they are used in our reference vehicle and hence depict a legit architecture for evaluation.

The *ECU Authentication Phase* was evaluated on a Texas Instruments Stellaris LM4F120 Launch-Pad Evaluation Board, leveraging the SRAM PUF accessible from prior work [77]. Given that most devices incorporate SRAM, an intrinsic SRAM PUF can be implemented on them. In cases where SRAM is unavailable, a PUF can also be implemented on DRAM or Flash memory.

Cryptographic operations are implemented using the BearSSL[1] library, specifically designed for small embedded devices with Elliptic Curve Cryptography (ECC) support. All cryptographic operations involving ECC are conducted on the NIST P-256 curve. Table 5.1 summarizes the essential characteristics of our evaluation networks. Despite CAN-FD supporting a bandwidth of up to 8 Mbit/s, our network yields a lower throughput. We attribute this deviation to the CAN controller of the PiCAN-FD shied.

Operation	**Classic CAN**	**CAN-FD**
Throughput (Mbit/s)	0.95	4.65
Round Trip Time (RTT) (ms)	0.17	0.15

Table 5.1: Throughput and round trip time for classical CAN and CAN-FD

The traffic of each scheme phase occurs through a designated high-priority CAN message type. Specifically, all messages exchanged within a given phase share the same CAN identifier. Message identifiers $0x1$, $0x2$, and $0x3$ are assigned to the traffic of the first, second, and last phases, respectively. This approach protects our scheme from potential delays caused by network congestion, as other messages are preempted by high-priority traffic.

Furthermore, we allocate the first two payload bits for the current communication step of each phase. Two bits suffice since there are at most four communication steps per phase. Additionally, we reserve five additional payload bits to address a specific $\mathcal{E}_\mathrm{i}$. Although theoretically, we can address 32 different ECUs, in an SDV, we expect far less ECUs connected to a single CAN. This expectation arises from the diminishing prevalence of large CAN networks in contemporary road vehicles.

Finally, we allocate four bits to track the fragmentation of primitives across different frames, indicating the order of these frames, as not all primitives can be encoded within a single frame. For example, an implicit certificate, with a size of 65 bytes, requires classical or two CAN-FD frames. Table 5.2 illustrates the byte requirements and the corresponding number of CAN frames for encoding key primitives in our scheme. This allocation results in the use of eleven payload bits for transmitting metadata, reducing the available payload to 53 bits (classical CAN) and 501 bits (CAN-FD).

[1] https://bearssl.org/

Primitive	Size (bytes)	#CAN frames	#CAN-FD frames
Request Frame	–	1	1
Public Key `CAN_QECU_i`	65	10	2
Implicit Certificate `CAN_IC_i`	65	10	2
`CAN_Kcanid`	16	3	1
Nonce	8	2	1

Table 5.2: Size and number of frames for scheme primitives

Runtime Measurements

After implementing the scheme, we investigate its timing performance. We differentiate between the initialization time and the time required to generate cryptographic keys for CAN message types. The *ECU Authentication Phase*, the *Certification Phase*, and step [A] of the *Key Computation Phase* are considered as the scheme initialization since they only need to be executed once during the lifetime of an implicit certificate.

The SRAM PUF results are derived from measurements taken in our prior works [77]. As the SRAM PUF was only available on the Stellaris devices used in the prior work, we implemented a dummy function on the ESP32-EVB and RSP3b+ boards to simulate key extraction from their SRAM.

To accurately measure the time consumption of the *ECU Authentication Phase*, we replaced the time consumption of the dummy function with values obtained from our previous works. The SRAM PUF response is accessible only after a device reboot, as it relies on the startup values of the SRAM cells. This aligns with our scheme, as we recommend running the entire scheme at vehicle startup. Given that reboot time varies across different ECUs, we excluded the actual reboot time from our measurements to prevent distorting results.

Figure 5.4 presents three measurements for the first two scheme phases, differing in the number of ECUs in use.

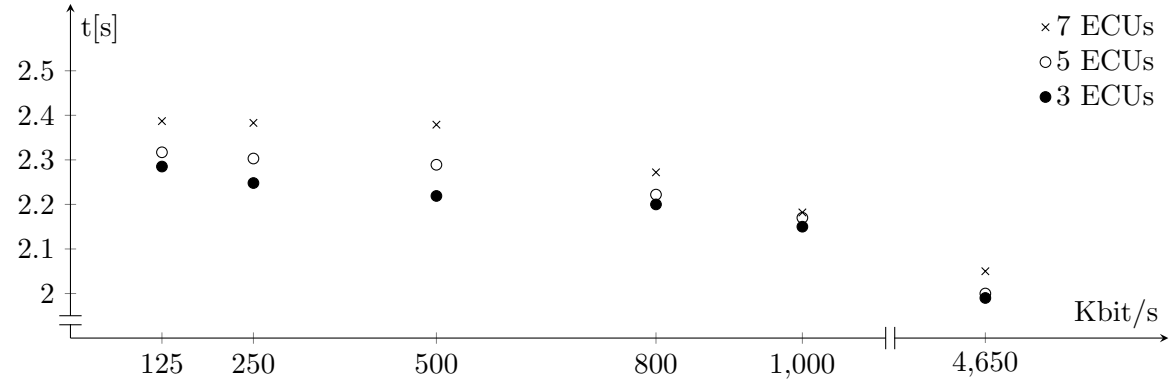

Figure 5.4: Time consumption of the first two scheme phases in a classical CAN network with varying numbers of ECUs and for various bandwidths. The network throughput does not have a significant influence, but rather the computational overhead.

Generally, more ECUs in the network leads to increased traffic and prolonged time for implicit certificate distribution. However, the bandwidth has a marginal impact on timing for a given number of ECUs because relatively little data needs to be transferred. We observe an average time of 2.3167s for authenticating devices using a PUF and distributing implicit certificates at

1 Mbit/s, the classical CAN bandwidth. This represents a reasonable overhead, considering initialization occurs only at vehicle startups.

In our second experiment, we measured the time required to establish a specific number of symmetric keys among a fixed number of ECUs, indicating the time overhead of the *Key Computation Phase*. We explored three network setups: We were initially connecting five ECUs, then adding two more, and finally extrapolating our results to simulate a CAN with 30 ECUs. We assumed each key is shared by all ECUs, acknowledging that this is a worst-case scenario as not every ECU subscribes to all CAN message types. Besides, as explained earlier, in such a case, an optimization would be to deliver only one key to all ECUs because the count of network participants remains identical.

Figure 5.5 illustrates our results on a classical CAN network with a bandwidth of approximately 1 Mbit/s. It took 27.25 milliseconds to establish a single shared key between seven ECUs and 135.75 milliseconds between 30 ECUs. The computational overhead is small compared to network transmission time, also because our implementation uses an OTP in the*Key Computation Phase*.

As depicted in Figure 5.5, both the number of subscribed ECUs and the number of keys have a linear impact on the time behavior. We do not get perfectly straight lines due to network collisions and packet loss, as we had to resend frames in such cases, resulting in larger time consumption.

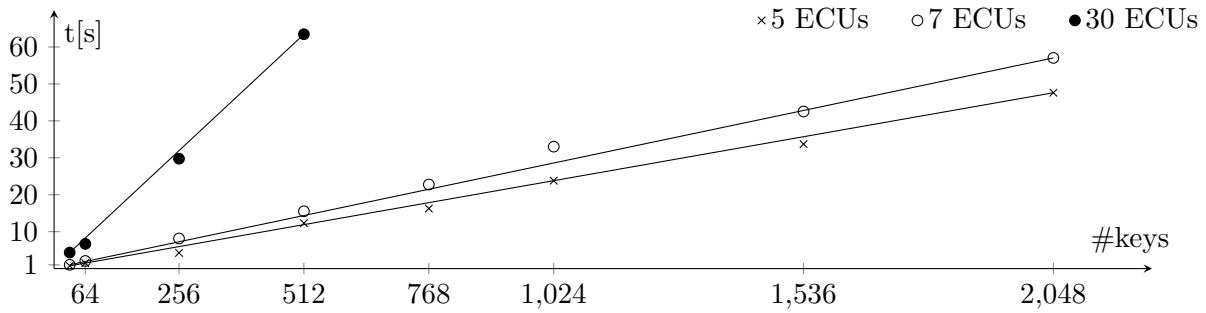

Figure 5.5: Time consumption on a classical CAN network at 1 Mbit/s for creating sets of group keys being shared by 5, 7, and 30 ECUs.

In scenarios where all ECUs share the same keys, time consumption increases dramatically. For instance, it takes roughly one minute to establish 512 symmetric keys in a bus topology. In addition to the increased time, the ECU would require at least $512 \cdot 16B = 8192B$ unused memory to store the keys. This time would be unacceptable in a real scenario, which, however, is unlikely to occur and only of theoretical interest.

Practical Implications As stated, we compute a key for each CAN message type in the third phase. In the classical CAN protocol, eleven bits are allocated for the message type within the arbitration frame field. Consequently, a maximum of $2^{11} = 2048$ keys is needed. However, CAN-FD utilizes an extended format with 29 bits reserved for arbitration, theoretically allowing for $2^{29} = 536,870,912$ different message types.

As maintaining such an extensive key set is impractical, we propose two options to reduce the potential number of keys if necessary: Firstly, the same key can be utilized for message types subscribed to by the same set of ECUs. In other words, each distinct group of ECUs shares

precisely one key. Secondly, a key is only necessary for safety-critical or confidential messages. A preliminary network analysis should identify which message types require security, determining the number of keys needed.

5.3.6 Sub-conclusion

An efficient key distribution protocol is essential in safeguarding signal-based communication against manipulation. Efficiently updating cryptographic keys is particularly vital for SDVs with a modular software architecture. The three-phased scheme presented here provides a partial solution to our research question RQ2: Cryptographic manipulation protection necessitates the availability of keys on the involved ECUs. The ability to update keys in a modular environment is essential to prevent key leakages and adequately react to system changes.

Instead of using the group key from the third scheme phase solely for securing messages of a specific CAN identifier, it could be employed to secure a *topic* in a service-oriented environment. This way, mapping a topic to a CAN message is easily achievable from a security perspective, facilitating the coexistence of legacy communication and novel Ethernet-based approaches.

Our key computation and distribution scheme leverages PUFs to verify the identity of ECUs and employ implicit certificates to establish a key pair on authenticated ECUs. As implicit certificates are small and demand low computational resources, they are particularly suitable for environments such as CAN networks with resource-constrained ECUs. To our knowledge, this has yet to be done in prior works. These lightweight certificates are ultimately used to compute a session key and obtain an authentic group key.

While the scheme initialization, comprising the first two phases, consumes on average 2.316 s in our exemplary CAN network, a 128-bit group key can be created and distributed in only 27.25 ms in the third phase. This phase can be repeated arbitrarily often as long as the implicit certificate is valid. We recommend executing the three phases at vehicle startup after significant changes (e.g., ECU replacement, firmware updates) or at predefined time intervals.

We are confident that implicit certificates also pave the way for deploying other automotive and security-demanding technologies, such as over-the-air updates or V2X communication.

5.4 Securing the FlexRay Protocol

The following sections are inspired by the research presented in the article “Securing FlexRay-based in-vehicle Network” [3], originally published in the Microprocessors and Microsystems journal in 2020.

In the following sections, we look at FlexRay, another legacy network entirely different from CAN. Instead of employing arbitration logic and allowing communication at any time, FlexRay has a deterministic communication schedule divided into time slots as outlined in Section 2.3. Although we assume that FlexRay will disappear from automobiles in the long term, especially from SDVs, our reference vehicle demonstrates that this technology may still persist in specific places for a while. For holistic protection, we explore this protocol and examine how communication can be made tamper-proof.

This objective directly results from the security requirement analysis conducted in Chapter 4, which highlights the importance of *System Integrity* in the zone containing the dynamic modules.

5.4.1 Security Requirements

To protect the FlexRay network against message spoofing, we leverage slot-based communication to compute and transmit authentication tags. We use the second FlexRay channel for a higher degree of fault-tolerance and security and propose means to efficiently maintain cryptographic keys, ensuring that key leakages or protocol updates can be seamlessly handled throughout the lifetime of road vehicles. Since FlexRay, despite limited attention, is employed for highly safety-critical communication, our solution contributes to designing a secure vehicle.

Addressing one of the most severe automotive security threats, the injection of forged commands [68], our primary security objective is the ability to verify the authenticity of FlexRay traffic. Besides authentication FlexRay traffic, Nilsson et al. [78] highlight encryption as another security property for FlexRay networks, confirming the earlier identified requirement for data confidentiality. However, as most traffic typically does not carry secret information but rather safety-critical data, confidentiality only plays a minor role in our work. We even argue that safety-critical data should not be encrypted due to potential decryption flaws, impeding vehicle safety. Nevertheless, we contend that our work can be readily adapted to achieve encryption, thereby potentially safeguarding passenger privacy [71].

5.4.2 Key Organization

Strong, fresh cryptographic keys and efficient data management methods are prerequisites for secure SDVs. Recent research has organized two primary organizational approaches for symmetric keys: The first method involves issuing a single key for a group of network nodes [47], while the second one associates keys with distinct message types [73].

Depending on the network setup, type-based keying techniques can result in an enormous number of keys due to the typically large number of potential message types. A purely recipient-based approach proves challenging since the recipient is often unknown to a sender node in a broadcast protocol. In response, we propose associating cryptographic keys with FlexRay time slots, encompassing static and dynamic slots. As illustrated in Figure 5.6, we differentiate between three possible options:

1. One key for all time slots.
2. One key for each time slot.
3. One key for a group of time slots.

While the first option offers simplicity, a key leakage will have significant consequences, as all nodes share the same key for cryptographic operations. Alternatively, each FlexRay time slot is associated with a key. Consequently, all nodes interested in a specific slot require the corresponding key. Since the communication schedule is predefined, a key establishment phase could run during the network's start. As a maximum number of 1023 static slots is theoretically possible, a FlexRay node still needs to administer 1023 keys in the worst case. This may be infeasible in practice due to constrained hardware and memory overhead. Therefore, we recommend the third option, which groups time slots in batches so that fewer cryptographic keys are required. In particular, if a specific FlexRay node occupies multiple successive slots, it may be reasonable to use the same key. The third option demands a thorough network design process, as the batch size depends on the network parameter and the exchanged data. When it comes to authentication,

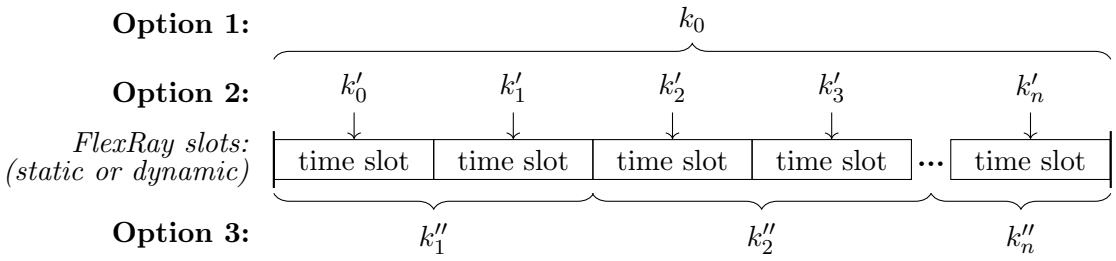

Figure 5.6: Either a single shared key is used for all slots (option 1), a unique key is created for each slot (option 2) or a series of slots (option 3) is associated with a key.

a verification delay must be acceptable if one authentication tag is computed for multiple slots. Our evaluation shows the negligible impact of such a delay from a safety perspective.

5.4.3 Traffic Authentication

In the following section, we propose means of transmitting authentication data over FlexRay and explore the security implications of leveraging the dual-channel model. Rather than introducing a new authentication scheme, we emphasize practical aspects, specifically organizing and updating cryptographic keys and transmitting authentication data over FlexRay. The ultimate goal is to provide data authenticity for both the static and the dynamic segments of a FlexRay communication cycle.

Although performing a successful spoofing attack on a static slot is technically challenging, the requirement for authenticated traffic is also valid within a static segment, as a resourceful attacker might attempt to prevent a benign node from sending a null frame in an empty static slot and subsequently inject fake messages.

Data authenticity typically involves using keyed hash functions and generating a MAC as an authentication tag. A key challenge in authenticating in-vehicle traffic lies in the efficient and reasonable transmission of these authentication tags, given that automotive legacy networks are typically bandwidth-constrained and must fulfill strict real-time requirements. In the subsequent discussion, we differentiate between the two FlexRay operating modes: single-channel and dual-channel mode.

Single-channel mode

The single-channel mode describes FlexRay communication over one physical link. We distinguish between two options on how to transfer an authentication tag. The first option describes the naive way of integrating a MAC into the frame that is being authenticated. This can be achieved by either appending it to the payload or embedding it into other frame fields, such as the header CRC field and the frame trailer. Both fields are used for error correction, an essential capability of a communication protocol. Related works have demonstrated how message authentication codes can be equipped with error-correcting capabilities [79]. This concept is relatively old but has again caught the attention of the 5G technology [80]. In total, a FlexRay frame provides $11 + 24 = 35$ bits for error correction. According to the birthday paradox, an attacker would observe, on average, the first collision after $2^{18} = 262,144$ authenticated messages, which is a reasonably poor security level. Frequent key updates could mitigate the effect of short MACs since it is harder to correlate collisions. Short MACs can also result from MAC truncation; a method to deal with MAC in case of resource limitations. For instance, AUTOSAR [19] pro-

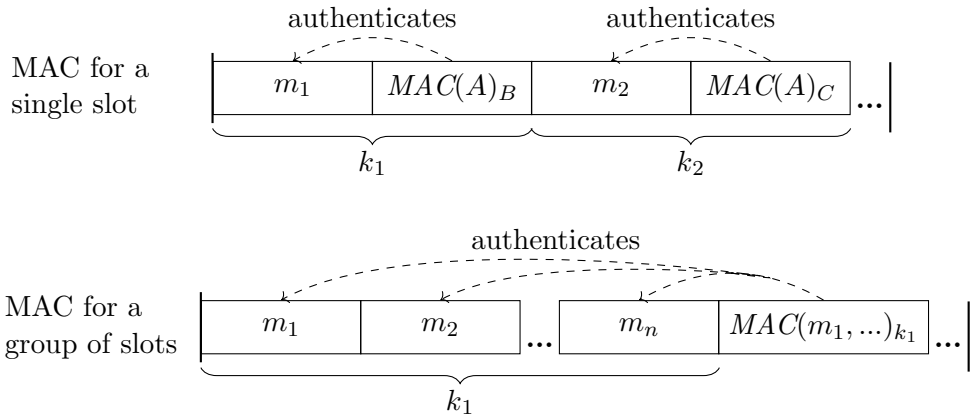

Figure 5.7: Authentication tags for groups of slots are stored in a dedicated time slot.

poses the use of MAC truncation. Besides the limited security properties, this option requires adaptions of the FlexRay controller because error correction typically happens below the network layer. Consequently, this approach is not a backward-compatible solution, which is why we generally discourage its realization, but we still mention it for completeness. Alternatively, the MAC can be appended to the actual frame payload. Our evaluation shows that this leads to a bandwidth reduction of roughly 3.15%.

The second, more convenient option is to reserve a dedicated time slot for transmitting a MAC. If the second keying option is selected and a unique key is associated with each slot, then two adjoining slots are considered one unit. The first slot transmits the payload, while the subsequent one delivers the authentication data. Depending on the network setup and the available hardware resources, the slot duration may need longer to compute a MAC. In such a case, we propose to compute an authentication tag for a group of consecutive slots and transmit it in a dedicated slot. This approach aligns with the third keying option, where one key is associated with multiple slots. Figure 5.7 illustrates both cases. This approach is fully backward compatible, although the bandwidth decreases up to 50% in the worst case, as our evaluation shows.

Dual-channel mode

The optional second FlexRay link enables concurrent data transmission. That way, a doubled bandwidth of at most 20 MBit/s is possible if both channels transmit distinct data. Alternatively, the second channel increases reliability by redundant data transmission. This is interesting from a safety point of view since data exchange remains possible even if one channel is faulty or completely broken. We suggest leveraging this second channel to transmit MACs efficiently. For this purpose, we again differentiate between two options: The first one uses the second FlexRay channel exclusively as a cryptochannel to not lower the bandwidth on the *data channel*. More precisely, the authentication tag of a given message is sent concurrently over the second channel to the receiver. While the static segments are synchronized on both channels, the dynamic ones may differ due to a varying slot size. These circumstances have to be taken into account by network developers. That means the layout of the dynamic segment should be the same to allow undelayed transmission of cryptographic and payload data. Consequently, the bandwidth on the data channel remains unaffected since additional cryptographic data is sent over the crypto channel. This option is suitable if the second channel is not required for (redundant) data transmission.

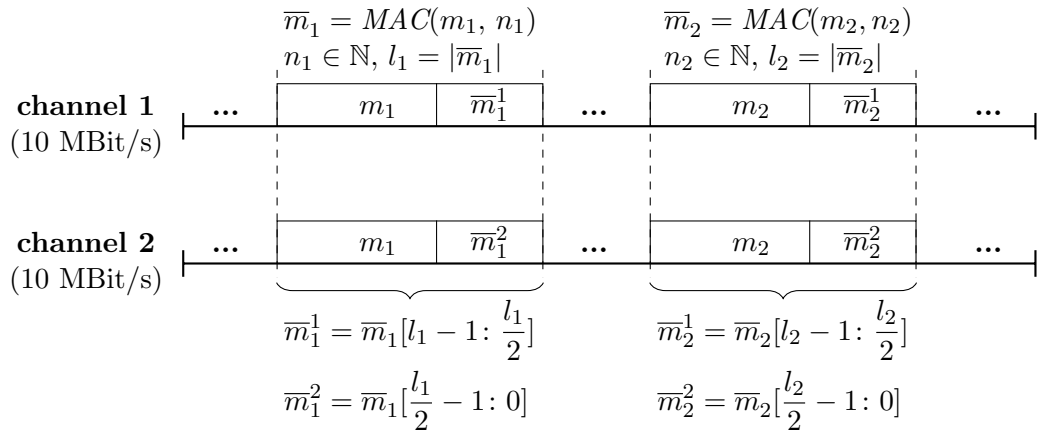

Figure 5.8: The MAC $\overline{m}_i$ is split into two equally sized parts $\overline{m}_i^1$ and $\overline{m}_i^2$, which are transmitted over distinct physical links. If one channel breaks, authentication is still possible at a lower security level.

The loss of redundancy, however, is the obvious downside of this option, as the second channel is wholly reserved for cryptographic data. Since redundancy may be a crucial safety prerequisite, our second option enables redundant payload transmission while cryptographic data is split on both channels. Let m be a data packet ready for transmission. The sender node first authenticates m by computing $\overline{m} = MAC(m, n)$, where n is a randomly chosen and publicly known nonce, that guarantees freshness. Subsequently, it splits $\overline{m}$ into two equally sized parts $\overline{m}^1$ and $\overline{m}^2$, where $\overline{m}^1 = \overline{m}[l - 1 : \frac{l}{2}]$ and $\overline{m}^2 = \overline{m}[\frac{l}{2} - 1 : 0]$ with l denoting the bitlength of $\overline{m}$. Eventually, the sender sends $(m, \overline{m}^1)$ over the first channel and $(m, \overline{m}^2)$ over the second one. The receiver obtains m on both channels and reconstructs the original MAC by concatenating $\overline{m}^1$ and $\overline{m}^2$, i.e., $\overline{m} = \overline{m}^1 \,||\, \overline{m}^2$. The receiver verifies $\overline{m}$ and distinguishes between three possible outcomes.

1. $\overline{m}$ can be successfully verified. The receiver accepts m and processes its content.

2. Only $\overline{m}_1$ or $\overline{m}_2$ can be successfully verified. There can be multiple reasons for this. Either the transmission was faulty, or one channel had been compromised. In that case, the receiver verifies $\overline{m}_1$ and $\overline{m}_2$ independently and accepts m if at least one MAC part is valid. Since the integrity of one channel cannot be guaranteed anymore, we suggest lowering the vehicle's driving capabilities as suggested in [9] and discussed in Section 5.4.6. To mitigate potential cryptanalysis attacks, the corresponding cryptographic key is updated according to the protocol in Section 5.4.4.

3. $\overline{m}$, $\overline{m}_1$ and $\overline{m}_2$ cannot be verified. In that case, we either assume severe network failures or an attacker who controls both channels. We suggest reporting a security incident, possibly leading to an emergency halt. To prevent an overreaction, such an incident must be analyzed regarding the expected damage. Chapter 8 presents one possibility to do so.

Since we accept the message in the second case, cryptanalysis attacks become easier if only half of the authentication tag can be verified. Therefore, we conduct a key update in the second case. We claim this approach complies with the earlier MAC truncation method, which has a similar effect to our MAC division technique as shown in Figure 5.8.

5.4.4 Secure Key Updates

A weak, broken, or leaked key can allow $\mathcal{A}dv$ to subvert security mechanisms and eventually gain unauthorized access to the vehicle. Automotive systems, which typically operate for many years, have an even higher demand for updating cryptographic keys.

A naive approach would involve computing a new key on a master node and then sending it encrypted to all recipient nodes. Alternatively, key updates can be organized in a distributed way where each node computes a new key once a predefined event is triggered. This event could be the reception of an authenticated trigger message or the elapsing of a predefined time. We propose deriving new keys from a locally maintained hash-based key chain in a *reversed* manner, meaning that the first key in use is the last chain element. When a key update is triggered, each affected node retrieves the *previous* chain element and adopts it as a new key. This approach allows the derivation of forward-secure keys, provided that the underlying hash function is pre-image resistant. More precisely, we assume a pre-shared key K shared among all benign FlexRay nodes $\mathcal{E}_\text{i}$. We acknowledge that legacy ECUs often lack tamper-proof memory where such a key could be securely stored. Still, we refer to PUFs as used earlier in this chapter, allowing us to retrieve a unique key from physical properties.

The master key K serves as a trust anchor between $\mathcal{E}_\text{C}$ and $\mathcal{E}_\text{i}$ and is used to encrypt confidential data. As illustrated in Figure 5.9, we distinguish between two phases: the *Seed Distribution Phase* and the *Key Update Phase*. k_c denotes the current key retrieved from the hash chain position $j \in [0, l-1]$ and is expected to be updated later.

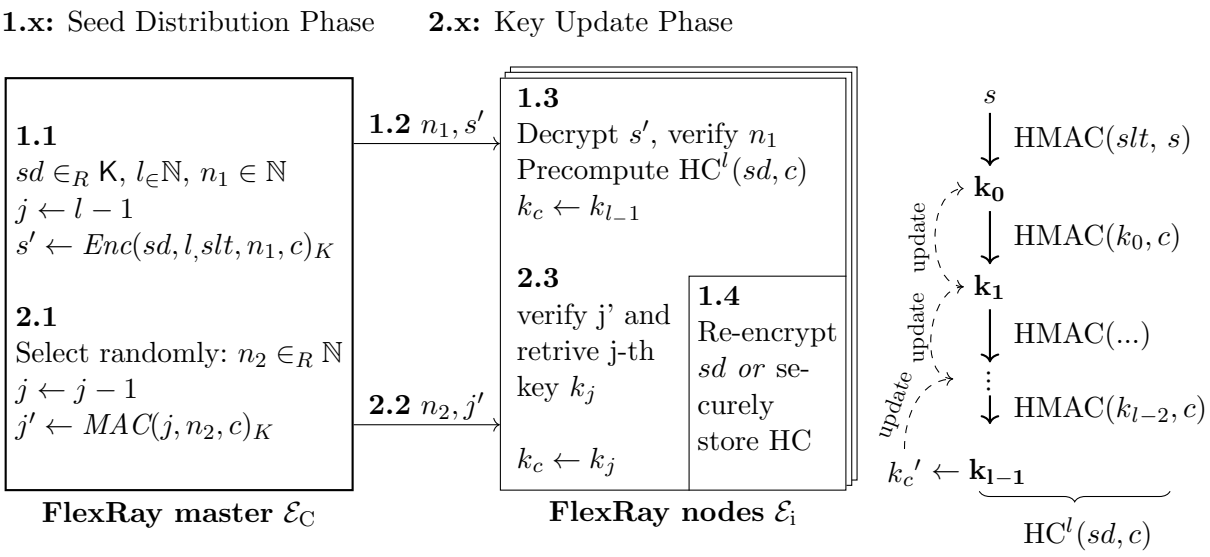

Figure 5.9: Each FlexRay $\mathcal{E}_\text{i}$ derives a hashed-based key chain FRHC of length l from a seed sd for a context string c. The current key $k_c{}'$ is initially set to the last chain element and moves backwards each time a key update is triggered.

Seed Distribution Phase

The *Seed Distribution Phase* starts (Step 1.1) with the master node $\mathcal{E}_\text{C}$ randomly selecting a seed $sd \in_R \mathsf{K}$, where K denotes the key space. It also selects a nonce $n_1 \in_R \mathbb{N}$, the chain length $l \in \mathbb{N}$, a random bitstring slt from a high-entropy source, and an optional context string c. The nonce n_1 ensures freshness and prevents replay attacks. Depending on the selected keying technique, a FlexRay network may need to maintain multiple keys simultaneously (e.g., for various slots).

In such cases, a context bitstring c allows differentiation between different keys while still using the same seed. $\mathcal{E}_\mathrm{C}$ encrypts sd, l, slt, c, and n_1 using K and publishes it to all FlexRay nodes $\mathcal{E}_\mathrm{i}$ (Step 1.2). $\mathcal{E}_\mathrm{i}$ decrypts the received data, verifies their freshness with the help of n_1, and initializes k_c with the last chain element k_{l-1} (Step 1.3). For this, the hash chain $\mathrm{HC}^l(sd, c)$ is recursively built, following an extract-and-expand paradigm. We use an HMAC Key Derivation Function [81] for the extraction and expansion stages.

Initially, a pseudorandom key k_0 is *extracted* from the seed sd as a fixed-length input. This is achieved by computing $k_0 = \mathit{HMAC}(slt, sd)$. Note that the secrecy of the entire hash chain relies solely on sd, emphasizing the need to protect sd against unauthorized access. In contrast, c and slt are public values, with the latter enhancing the strength of $\mathrm{HC}^l(sd, c)$. The salt supports "source-independent" extraction [81] and makes the output more similar to a random oracle answer.

Next, k_0 is *extended* to a chain of pseudorandom keys through $\mathrm{HC}^l(sd, c)$. To achieve this, we compute $k_j = \mathrm{HMAC}(k_{j-1}, c)$, $j \in [1, l-1]$.

The key chain $\mathrm{HC}^l(sd, c)$ can be maintained in two ways. One approach involves $\mathcal{E}_\mathrm{i}$ unrolling the entire chain and storing each precomputed key encrypted in memory. While this method leads to higher memory consumption and, therefore, is often infeasible in practice, it enables faster key updates since the corresponding key only needs to be decrypted. Alternatively, only the seed is stored encrypted, and key updates are computed on the fly.

When opting for the latter approach, a key update from the j-th to the (j-1)-th position necessitates j calls to HMAC because $\mathrm{HC}^l(sd, c)$ has to be traversed in reverse. We assess computational and memory overhead in Section 5.4.7.

Key Update Phase

Once the *Seed Distribution Phase* has terminated, a key update can be efficiently accomplished without causing significant network overhead. A fresh key is selected by moving back one step in the hash chain. Specifically, $\mathcal{E}_\mathrm{i}$ overwrites k_c with its predecessor in $\mathrm{HC}^l(sd, c)$. During a key update, k_c is shifted from k_j to k_{j-1}. This reverse mechanism ensures forward secrecy if the underlying cryptographic hash function is considered secure, meaning the pre-image computation is infeasible. Consequently, an adversary gains no information about subsequent keys, even if a key at a specific position has been revealed.

The master node $\mathcal{E}_\mathrm{C}$ triggers a key update by first decrementing the current chain position j by one. Then, it authenticates the new value of j, resulting in j' (Step 2.1). The master includes an optional context string and a nonce n_2 to j' to guarantee freshness. Otherwise, an attacker could replay an old j and force the production of an (old) key in a given communication cycle.

$\mathcal{E}_\mathrm{C}$ subsequently distributes the key update trigger message j' to the FlexRay nodes $\mathcal{E}_\mathrm{i}$ (Step 2.2.). The new chain position is ambiguous information for $\mathcal{E}_\mathrm{i}$ because a fresh key is always obtained by moving back exactly one position in the hash chain. This ambiguity, however, contributes to protocol synchronization between the nodes. Alternatively, the *Key Update Phase* could be executed after some time, e.g., after a predefined number of cycles, since FlexRay offers strong synchronization, making such a protocol modification easily realizable.

Upon receiving j', $\mathcal{E}_\mathrm{i}$ verifies its authenticity and freshness and obtains the new key from $\mathrm{HC}^l(sd, c)$ (Step 2.3). This can be achieved in two ways: If $\mathrm{HC}^l(sd, c)$ has been precomputed

during the *Seed Distribution Phase*, the new key can be instantly updated by reading and decrypting it from memory. Alternatively, $\mathcal{E}_\mathrm{i}$ has to perform $j+1$ recursive hash operations on the seed to obtain k_j. Once the *Key Update Phase* has been repeated $l-1$ times, $\mathcal{E}_\mathrm{C}$ has to rerun the *Seed Distribution Phase* and distribute a new secret sd, as all keys of $\mathrm{HC}^l(sd, c)$ have been used up.

Automotive sessions

The presented key update protocol enables the fast renewal of forward-secure keys without long key negotiations. We recommend defining automotive sessions as a known concept from related works [43, 73, 82]. Given the strong synchronization between FlexRay nodes, a promising approach involves combining event-driven and time-driven session strategies. An event-driven session prompts a key update after a specific event (e.g., a software update or workshop repair). In contrast, time-driven sessions initiate the key renewal process after a predefined duration (e.g., after being parked for at least a week). In the context of our reference vehicle, a new session might begin whenever the operating mode changes, as is the case when switching from automated to remote driving.

5.4.5 Realization in FlexRay

A key update can be considered an exceptional event, requiring sporadic messaging. Consequently, we propose handling key renewals within the optional dynamic segment, which offers event-driven communication. Recall that the dynamic segment allows for the exchange of variable-length and priority messages, while the static segment is designed for periodic real-time data transmission in fixed-length time slots.

To implement this, the communication schedule should be structured so that the master node transmits a key update trigger message at the highest priority, ensuring it cannot be suppressed by any other message. That means the first dynamic slot is assigned to the master. From a technical standpoint, the smallest slot counter value should be associated with the master node, reserving the initial mini-slots for a key update. Other nodes can occupy subsequent dynamic slots to broadcast messages if no update is due.

5.4.6 Security Discussion

The techniques presented for traffic authentication on the FlexRay bus rely on the unforgeability of MACs under a chosen message. Successful authentication should only be possible for ECUs possessing the correct key, even if an adversary gains access to valid (data, MAC) pairs. AUTOSAR recommends a default key length of 128 bits as successful brute-force attacks become extremely unlikely. The minimum MAC length should also be at least 64 bits.

According to the birthday paradox, an attacker observes a collision after $2^{n/2}$ messages on average. While a collision does not immediately lead to the ability to forge MACs, it gives the attacker an advantage in cryptanalysis. Hence, the secure construction and application of message authentication algorithms are crucial since any flaw can compromise the system, even if the underlying cryptographic primitive is proven secure. For example, the secret prefix method [83] exploits the simple and insecure construction $MAC(m)_k = Hash(k||m)$, where a secret k is prepended to a message m.

In combination with an iterative method such as Merkle-Damgård, length extension attacks enable the construction of valid MACs for arbitrary messages that have $MAC(m)_k$ as a prefix, although k remains unknown to the adversary. In contrast, a CBC-MAC construction is generally secure, but its improper application can lead to an existential forgery. Specifically, CBC-MAC must only be used for messages of fixed length; otherwise, an adversary can construct an unseen message for a given (message, MAC) pair.

In our work, we find the HMAC construction promising for two reasons. First, it is not prone to the aforementioned length extension attack due to its internal structure of an inner and outer hash. While the inner hash includes the actual message and may be of variable length, the input of the outer hash always has a fixed length, preventing length extension attacks. Despite two calls to the underlying hash function, an HMAC is still efficient because the outer hash is only computed over a short two-block message. Second, HMAC demands weaker security assumptions specifically suitable for a long-term operating environment. Only weak collision resistance must be guaranteed for the underlying hash function [84]. For this reason, even HMAC-MD5 is secure despite the fact that the hash function of MD5 is considered broken.

The presented scheme for key updates relies on the secrecy of the initially distributed seed that is only required to retrieve a new key from the underlying key chain. In the meantime, we suggest keeping it encrypted in the available memory. We utilize the root key for its encryption, assuming it is securely stored in hardware or can be retrieved from unique physical properties as we did before leveraging a PUF. Session keys can be buffered unencrypted in the same memory, assuming an attacker without access to the memory while a session is running.

In Section 5.4.3, we suggest splitting authentication tags into equally sized parts sent independently over both physical channels. This approach lowers the security level, as we accept the corresponding message if at least one tag part can be verified. Consequently, an attacker requires fewer resources to undermine security since only half of the MAC bits need to be attacked. Our experiments revealed a faulty transmission rate of 0.25% in our exemplary FlexRay network, meaning an unverifiable MAC is not necessarily caused by an attack.

Therefore, instant rejection of any unverifiable message may lead to driving instabilities if they cannot be assessed adequately. The dual-channel mode allows us to weigh potential security incidents and react appropriately if authenticity checks fail on a single physical link. Specifically, we suggest informing a monitoring system in such a case, which constantly supervises and assesses the vehicle's capabilities, i.e., the current vehicle state. Regarding our reference vehicle, this could be the self-perception system employed in the reference vehicle, enabling continuous monitoring of the vehicle's security and safety state. Section 8 explains what appropriate measures a vehicle may select depending on its context and the perceived security flaw.

Key Lifetime

The presented key update scheme details how to update a key but does not state when an update should happen. The National Institute of Standards and Technology (NIST) notes in its *Recommendation for Key Management* [85] that short cryptoperiods generally enhance security unless a system's paramount concern is denial-of-service attacks.

A reasonable key lifetime depends on various risk factors, such as the key's purpose, the strength of the deployed cryptographic algorithms, potential side channels, the operating environment, the number of stakeholders, data sensitivity, and the key derivation process. While each one has

to be individually assessed for a given system, all cryptographic keys should at least be renewed once the vehicle starts. In the following, we outline the main aspects of this assumption.

Lifetime of Road Vehicles Road vehicles typically have a long average lifetime spanning many years. Consequently, there can be a significant delay between when a key begins to be used and when an attacker obtains it. Adversaries may gain key access by subverting an in-vehicle network, exploiting side channels such as power supply analysis [86] that penetrate a system to steal sensitive data and circumvent cryptographic protection.

Due to road vehicles' expected long operating lifetime, even a small bandwidth would not pose a major problem for $\mathcal{A}dv$, given ample time to conduct an attack. Moreover, key updates mitigate the impact of increasing computational power available to the attacker. Road vehicles are often left unsupervised, allowing $\mathcal{A}dv$ to gain physical access and steal cryptographic keys. Therefore, a short crypto period effectively mitigates the impact of long vehicle lifetimes.

Operating Environment According to the NIST recommendations, a shorter crypto period is desirable for protecting critical data, as typically is the case in FlexRay networks. Another reason for short crypto periods is that the design and construction process of road vehicles typically involve numerous stakeholders, resulting in a higher probability of key leakages. Moreover, the complexity of supply chains means that key leakages may remain undetected for longer if the reporting of such incidents is not properly handled. Note that we require a uniform security incident reporting mechanism as part of the requirements analysis to prevent such scenarios. Therefore, any system change should lead to the renewal of cryptographic keys, particularly in times of SDVs incorporating a modular and updatable software architecture.

ECU Resources Legacy ECUs are usually constrained in resource, lacking tamper-proof memory and access to a high-entropy source. For example, a missing high-entropy source can lead to weak cryptographic parameters (e.g., initialization vectors, nonces), making cryptography more susceptible to being easily compromised. A short crypto period is a mitigation strategy for such resource limitations because it gives adversaries less time for a successful attack. Combining the arguments for frequent key updates with the NIST recommendation that symmetric keys should not be used after the end of the originator-usage period, we conclude that the renewal of cryptographic keys should happen at least at vehicle startup.

5.4.7 Evaluation

We used the NXP S12XFSTARTERKITE evaluation kit as a target platform for our implementation. The kit comprises two FlexRay nodes, each with a 16-bit MCU (S12XF512) and an XGATE co-processor. Specifically designed for autonomous high-speed data processing, the XGATE module aims to reduce the S12X interrupt load. According to the NXP documentation, the FlexRay nodes in this kit are commonly employed in applications such as active braking, lane departure warning, and driving assistance features. Each node has 512 KB of flash memory and 32 KB of RAM.

The evaluation hardware conforms to the FlexRay V2.1 specification and supports the data transmission over two channels. Table 5.3 shows the main network parameters that remained unchanged throughout our experiments.

Network Parameter	Value
Duration of macrotick (MT)	1µs
Length of communication cycle	5000 MT
Length of static segment	3000 MT
Number of static slots	75
Static slot duration	0.04 ms
Length of dynamic segment	880 MT
Duration of a minislot	40 MT
Average frame transmission time	66.78 µs
Average of erroneous transmissions	0.25%

Table 5.3: Parameters of our evaluation network containing two NXP S12XF512 nodes.

For the following experiments, we measure the average of 100 executions to obtain representative results. The number of slots and the payload only have a negligible impact on the transmission time. In our observations, when transmitting payloads of sizes 16B, 64B, 128B, and 254B, we noted a standard deviation of 0.392 µs. A total of 0.25% of the transmissions were faulty, meaning data contained errors after reception. The subsequent experimental results provide insights into the memory and computational overhead of the presented security concepts. The authentication algorithm employed was HMAC-SHA2-256, which was implemented in software using the BearSSL library again.

Network Overhead

Figure 5.10 shows the timing overhead imposed by the presented techniques for transmitting an authentication tag over FlexRay. The authentication tag is either integrated into the frame to authenticate its payload, or a dedicated time slot is reserved for the tag to authenticate a group of preceding slots. The baseline for comparison is unauthenticated FlexRay communication.

We measured the time it takes until a data set of varying sizes is available at the receiving node. That means the elapsed time also includes the fragmentation time on both the sender and receiver sides. This experiment assumes a maximum load, so the test data was split into 254 bytes (i.e., the maximum frame payload).

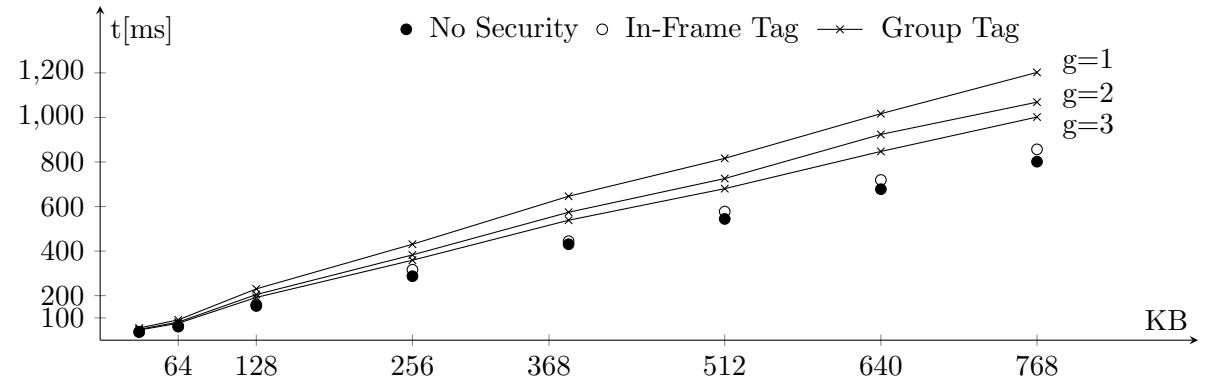

Figure 5.10: Network overhead caused by the transmission of an authentication tag. The best solution is to append such tag to each frame, which, however, may not be feasible without hardware acceleration due to the small slot duration. The group size g indicates how many slots are authenticated by a single tag, which is included in a dedicated slot. All measurements are averaged values over 100 executions.

In our evaluation setup, we achieved an actual bit rate of 8.3 MBit/s, slightly below the maximum rate of 10 Mbit/s. This deviation is likely attributed to an unbalanced workload because the experiment was conducted in the static segment while the dynamic one remained untouched. Considering that a FlexRay frame allows a maximum payload of 254 bytes, the overhead to include an authentication tag of 8 bytes into each frame is minimal.

If we apply the dual-channel mode, the tag length is halved, as it is partially transmitted over both channels. In contrast, the bit rate decreases by roughly 50% if authentication tags are transmitted in a dedicated slot with a group size of 1. Recall that the group size indicates how many slots are authenticated by one authentication tag. A group size of 2 leads to a bandwidth reduction of roughly one-third, while a group size of 3 results in a bandwidth reduction of 25%. As the receiver node has to wait until the arrival of all data for verification, a larger group size leads to larger verification delays. Such a delay must be considered from a safety perspective, as delayed processing of braking commands, for instance, may cause passenger harm. A worst-case delay of 3 ms, the total duration of the static segment, would result in a path extension of roughly 0.09m if the vehicle is moving at 120 km/hr. Hence, the selection of the group size depends not only on the available computational power but also on potential safety implications.

From a pure network perspective, it is advisable to append a MAC to the payload when applying the single-channel mode. However, our computational analysis demonstrates that this solution may not be feasible on resource-constrained devices because the computation and verification of an HMAC can last longer than a slot. Thus, slot-based authentication can only be realized if enough computational power is available, enabling the computation of authentication tags in microseconds. We recommend using the dual-channel mode and splitting the authentication tag for fault tolerance, as explained in Section 5.4.3.

Computational and Memory Overhead

As mentioned, we assume a fixed key length of 128 bits and a payload of 254 bytes, with 8 bytes reserved for an authentication tag. We define the authentication tag as the highest 64 bits of the HMAC-SHA2-256 output. In total, it takes 15.49 ms on the FlexRay nodes to compute an authentication tag for random data, which far exceeds an acceptable overhead. Note that the duration of a static slot is only 0.04 ms.

We argue, however, that this limitation is specific to our setup rather than a general problem. Depending on available resources, an efficient implementation of an HMAC can be achieved, allowing for a time consumption of 20 µs [87] or even 2 µs [88]. Compared to our results on the S12X processor, the same implementation yields an execution time of 10.01 ms on a RPi 3B+. Without sufficient computational resources, slot-based authentication with a low group size appears unrealistic. Instead, either network parameters need adjustment, or the group size must be increased to enable computation and verification within the scheduled time slots.

Key Updates In the following, we investigate the presented key update mechanism from Section 5.4.4, which allows us to obtain forward-secure keys through hash chains. The master node has to rerun the *Seed Distribution Phase* once all keys have been used up, indicating that the chain has been fully traversed. In that way, a new seed is distributed, and a new key chain can be established on each node.

The subsequent experiment assumes two daily vehicle starts on average, yielding roughly 730 yearly key updates. This assumption aligns with our discussion in Section 5.4.6 to conduct a key

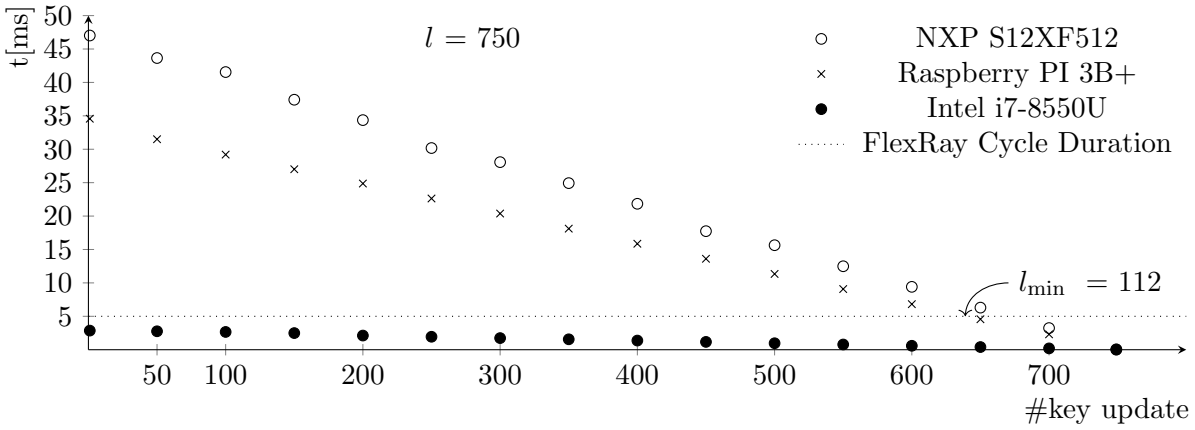

Figure 5.11: Execution time to perform the i-th key update. The i-th update requires $l - i + 1$ HMAC operations, with l denoting the length of the key chain. All measurements are averaged values over 100 executions.

update once the vehicle is turned on. Since these assumptions are based on average numbers, we define a slightly larger chain length of $l = 750$.

Considering each key occupies 16 bytes, the total size of $\mathrm{HC}^l(sd, c)$ is $16\mathrm{B} \cdot 750 = 12\mathrm{KB}$ if precomputed and securely stored. Many resource-constrained devices, like the deployed FlexRay evaluation boards, provide sufficient memory (512KB) for storing such a chain. However, the dynamic modules used in our reference vehicle do not possess an additional 12 KB for security tokens, meaning the hash chain must be shorter, or intermediate keys cannot be persistently stored. In a precomputed chain, each key must be encrypted with the root key to protect it against leakages. Consequently, a key update would result in a single decryption operation of the corresponding encrypted key.

Alternatively, instead of precomputing the entire chain, an ECU can compute the update key value on the fly. Once an update is triggered, each ECU builds up the chain using the seed and retrieves the key at the new position. As the key chain is traversed in reversed order, the first key update consumes most of the time, while the last one requires only a single MAC operation. Figure 5.11 visualizes the time consumption for the i-th key update. For comparison, we executed on an Intel i7-8550U processor, assuming that similar processors may be deployed in HPC platforms of SDVs.

The dotted horizontal line illustrates the 5 ms duration of a communication cycle in our setup. Consequently, achieving a key update within one cycle is feasible on the S12XF512 FlexRay node only if the chain length remains below $l_{\min} = 112$. Otherwise, it takes more than 5 ms to iterate through the key chain because a longer chain leads to more recursive computations and, thus, to a longer execution time. Establishing the full chain of length 750 consumes 47.02 ms on the S12X FlexRay nodes, which is a reasonable time in case the vehicle is not moving.

As expected, both the RPi and the i7-8550U outperform the FlexRay node due to their superior computational power. From a computational standpoint, the selection of the chain length must account for the available resources. Our key update protocol enables the renewal of a key within a single communication cycle, which is beneficial for maintaining synchronization among the nodes. We recommend retrieving a new key from the chain on the fly when needed rather than precalculating the entire chain in advance.

Combining the computational and network overhead results, we find group-based slot authentication most promising for FlexRay networks, as it leads to an acceptable computational overhead on both strong and resource-constrained devices.

5.4.8 Outlook

Our security solutions protect against message spoofing and forgery attacks on the FlexRay bus and present a possible way to update cryptographic keys. Nevertheless, the challenge of DoS remains a concern for FlexRay networks, where an adversary can disrupt communication by sending dominant bus signals. This straightforward attack could have severe consequences, particularly if safety-critical control units are impeded from exchanging messages. Since cryptographic solutions are ineffective in preventing this attack, the initial line of defense involves physical protection measures.

Moreover, the dual-channel mode, allowing for two physical links, adds an additional layer of security as both channels must be compromised for a successful DoS attack. As highlighted in our security requirement analysis in Chapter 4, addressing protection against DoS attacks is crucial, but this aspect is left for future work at this place.

Another potential threat is the hijacking of a FlexRay network by a remote attacker installing unauthorized software on an ECU through a compromised software update process. While our attacker model outlined in Section 5.1 does not attribute such capabilities to the attacker, we specifically address this issue in Section 7.

5.4.9 Sub-conclusion

In the preceding sections, we introduced methods for securing FlexRay-based communication against manipulation attacks and updating cryptographic keys. We recommended reversing hash-based key chains based on an initially distributed seed seed to maintain secrecy. The FlexRay nodes in our test network could securely update a key in less than 5 ms. Additionally, we exploited the dual-channel mode in FlexRay networks to transmit authentication tags, primarily to sustain a high level of fault tolerance. Specifically, we divided authentication tags and transmitted them over both physical channels. If one channel becomes corrupted, we can still receive data and verify its authenticity. As for the transmission of authentication tags, network capacity is not a limiting factor due to the relatively large frame size. However, our evaluation reveals that single-slot authentication is challenging for weak computational devices without crypto extensions.

Combining these recent findings, we conclude that safeguarding signal-based protocols, including CAN and FlexRay, necessitates precise planning of the legacy network. Due to often limited resources, the manufacturer must determine which data requires cryptographic protection, which hardware requirements are available (e.g., crypto extensions), and how cryptographic keys can be managed over a long period. In this chapter, we presented various strategies for securing both CAN and FlexRay communication, thereby offering an answer to research question RQ2.

6. Securing Service-Oriented Architectures

After examining the security of automotive legacy protocols in the last chapter, we now turn to novel service-oriented software architectures. In comparison to highly integrated systems that communicate with each other in a signal-oriented manner, only appearing in isolated places in SDVs and likely to disappear completely over time, modern software architectures are primarily designed to be modular, interchangeable, and updatable. They facilitate shorter development cycles, the addition or removal of features, and a high level of interconnectivity of the entire vehicle with its environment, enabling new mobility concepts.

The trend toward modular architectures, whether at the hardware or software level, has resulted, among other things, in AUTOSAR Adaptive. This standardized vehicle platform exists in parallel with AUTOSAR Classic and enables the development of complex, high-performance software components. AUTOSAR Adaptive, for example, provides a service-oriented runtime environment and supports different communication mechanisms such as SOME/IP. However, criticism has arisen due to its high level of complexity, partly inaccurate documentation, and the closed-source nature. Thus, the effort to create a standardized vehicle platform has yet to be fully successful. Instead, considerable efforts are underway to develop alternative solutions and tap into a significant market potential. Well-known OEMs are developing their software solutions, newcomers like ApexAI are entering the automotive market, and government-funded projects like UNICARagil contribute to designing vehicle software architectures with their middleware solutions.

Automotive middlewares, often called automotive OS, address numerous aspects, including the orchestration of software components, the optimal utilization of resources, the abstraction between hardware and software, and a uniform maintenance process. The latter, in particular, is essential for practical use: subsequent changes to SDVs must occur seamlessly, quickly, and securely without causing high costs or even recalls.

To ensure the highest possible flexibility, automotive middlewares are usually service-oriented. Consequently, the resulting software architecture is not static but rather dynamic. However, this dynamic nature increases the attack surface due to more external interfaces and more room for manipulation. Therefore, security is an integral component, as with signal-oriented communication. In addition to effective protection mechanisms, a security process is necessary to ensure that changes to the software architecture do not compromise security. Such a process includes, for example, the system description from a security perspective and the management and deployment of security tokens within the vehicle.

In this chapter, we address research question RQ3 by developing a security process for the ASOA [20] that ensures secure communication between services and considers the strict separation of system and functional knowledge. The ASOA is a software architecture for embedded

systems and HPC platforms developed for our reference vehicles as part of the UNICARagil project. Our security solution for the ASOA has been fully implemented and integrated into all reference vehicles.

The following chapter is structured as follows: First, we explain how the ASOA works in Section 6.1. We then propose a security process for the ASOA in Section 6.2, verify essential parts formally in Section 6.3, and eventually evaluate the runtime behavior in Section 6.4.

The following work is based on the publication "A Security Process for the Automotive Service-Oriented Software Architecture" [4], which was published in the IEEE Transactions on Vehicular Technology in 2023. I extend special thanks to Florian Frank, Wuhao Liu, and Marion Christl for their support in implementing the ASOA Security Process.

6.1 Automotive Service-Oriented Software Architecture

The Automotive Service-Oriented Architecture (ASOA) [20] is a framework designed to develop real-time, high-performance, and scalable software for an automotive ecosystem. One of its core elements is a middleware allowing for seamless and reliable data transmission between functional components, employing the DDS [27]. Notably, this middleware incorporates mechanisms to ensure strict adherence to timing requirements, particularly for safety-critical tasks, while also enabling operators to monitor event chains within vehicles. The ASOA also encompasses a comprehensive web-based tool for describing vehicular software architectures, specifically tailored to support the specification of communication endpoints, timing, and hardware requirements. Combining an architecture specification tool and a middleware that enforces predefined system parameters renders the ASOA a powerful framework for designing modern automotive systems, especially those reliant on zonal E/E architectures. Instead of deploying many ECUs with dedicated functions, zonal vehicles consist of fewer, yet more powerful, control units capable of providing computational resources for various, not necessarily related, functions.

6.1.1 Services

A service is considered a self-contained functional unit that interacts with other services by exchanging data. Depending on the resource requirements and topological constraints, a service can be deployed to any available control unit. The ASOA framework supports powerful, general-purpose control units and resource-constrained embedded systems, facilitating service deployment throughout the vehicle. Moreover, even external software components such as a traffic server or cloud functions can be represented as services, gradually dissolving the vehicle border to remote units. For instance, a service providing traffic information will likely run in the cloud but may still be connected to a route planning service inside the vehicle. Services, akin to Android applications, exhibit a lifecycle and can be started, paused, and terminated.

The ASOA framework emphasizes the complete decoupling of services from one another by excluding any global system knowledge, such as topological or architectural information. This design approach enables services to be easily relocated, divided, and maintained without modifying other automotive components. In contrast, SOME/IP services often possess an awareness of the existence of other services, e.g., from which they can query data. Consequently, system adaptions become more expensive when services originate from different stakeholders. An ASOA service may comprise multiple tasks, which can be scheduled based on their criticality by the underlying OS.

6.1.2 Communication

ASOA services adhere to a data-centric communication approach, wherein the architecture tool mentioned earlier empowers developers to define a service's communication endpoints and their corresponding quality expectations through a contractual mechanism that ensures compatibility. These contracts contain information concerning data formats, data quality attributes, and the frequency of data provision or requirements. Their purpose is to determine whether service endpoints can be connected, which is the case if their data type and quality parameters match. An endpoint denotes a service's input or output, referred to as a *requirement* or *guarantee*, respectively. A requirement can be *wired* to one or multiple guarantees, thereby enabling the receipt of data from the latter. Similarly, a guarantee publishes data on a specific topic to which an indefinite number of requirements can subscribe. Hence, ASOA services employ a publish-subscribe pattern for data exchange, obviating the need for services to possess explicit knowledge of their communication partners. A service is agnostic regarding the data source as long as the contractual obligations are upheld, i.e., the expected data type and quality specification are met. This design philosophy leads to a highly modular and decoupled automotive software system, wherein ASOA services are autonomous and independent entities. The ASOA framework leverages the DDS technology to map guarantees and requirements onto DDS endpoints, resulting in brokerless data transmission once services are entirely wired. However, the ASOA cannot be classified as decentralized because it incorporates a central communication management unit known as the *orchestrator*.

6.1.3 Orchestration

As explained, services do not carry information about their communication partners, necessitating external assistance for establishing connections. The orchestrator takes over this task by *wiring* ASOA services and controls the dataflows between them. Moreover, it manages the lifecycle of services by starting, stopping, and pausing them. In other words, the orchestrator connects services based on the prevailing operating mode of the vehicle and subsequently launches them. The operation mode indicates a vehicle's state, such as whether it drives automated or is under human control. In the former scenario, a service responsible for actuating the wheels might receive torques from a controller that converts a computed trajectory into low-end driving instructions. Conversely, the same service may obtain data from a user input device like a steering wheel. In both operating modes, the actuating services remain oblivious to the identity of the data provider, relying instead on the orchestrator's wiring configuration. The orchestrator learns from the architecture tool which services must be connected for a given operating mode.

The ASOA framework even allows for dynamic re-wiring of services during vehicle operation. For example, if a particular service cannot guarantee the quality of its output, it can be replaced by a more reliable source through reconfiguration. Figure 6.1 shows the orchestrator's role in a simplified ASOA system featuring four services. Both rows depict distinct service wiring, each representing an operation mode. In the first row, the orchestrator establishes a connection between a guarantee of Service 1 and a requirement of Service 3. In the second row, the same requirement of Service 3 receives the data from Service 2, e.g., due to a degeneration of Service 1. Here, Service 2 replaces Service 1 as the data provider.

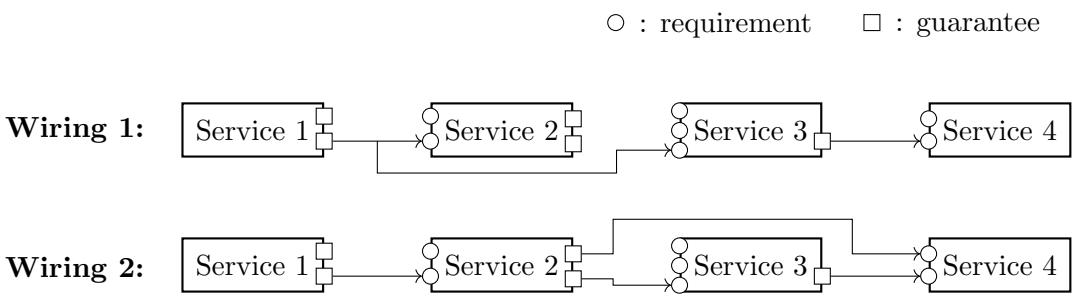

Figure 6.1: In the top row, the orchestrator wires the output (guarantee) of Service 1 with the input (requirement) of Service 3. In the bottom row, Service 2 replaces Service 1 as the data provider, resulting in a rewiring where it now provides data to Service 3. This dynamic rewiring capability enables adaptive data flows within the ASOA system.

6.1.4 Resource Utilization

In addition to facilitating service development and communication, the ASOA framework allows for resource optimization and analysis. A vital aspect of this endeavor involves deploying services to control units optimally regarding effective resource utilization. The framework solves an optimization problem to achieve this objective, accounting for topological constraints, processor capabilities, memory availability, and network resources. By considering these factors holistically, the ASOA framework strives to yield a balance in resource utilization. Furthermore, the ASOA enables the analysis of complex event chains, benefitting from the well-defined service internals in the architecture tool. The predictability and clarity of service behavior at runtime are crucial to automotive systems, particularly when deploying multiple software components on a shared platform.

6.1.5 Comparison

Unlike related solutions, the ASOA framework covers many aspects of the design and maintenance of automotive software systems. Its distinguishing characteristic lies in the architecture design tool, which enables the specification of crucial system parameters that are subsequently enforced by the ASOA runtime core. Currently, the ASOA uses DDS for data exchange; however, it will likely expand to support other communication middlewares. While ASOA services reveal internals like task allocation, related systems like ROS2 nodes are black boxes, making the retrieval of analytical information about node interferences hard. The central orchestration of services enables the design of isolated services that do not carry system knowledge, making it easy to replace and update them.

6.2 ASOA Security Process

The deployment of the ASOA framework in SDVs would currently lead to an insecure and unsafe system due to a missing security model. For instance, an adversary could easily impersonate the orchestrator and provoke a wrong wiring of services. Also, safety-critical control commands could be manipulated, and sensitive data would not be protected against unauthorized access. Therefore, we present a security process for the ASOA framework in this section, allowing vehicle operators to enforce legitimate dataflows while defending against typical manipulation attacks that have been carried out in prominent examples [18, 34, 89].

Although the ASOA uses DDS for communication between services and despite the effective security provided by the DDS Security Standard [30], we have intentionally opted against its selection as a security solution for the ASOA framework, primarily because the vehicle's security specification would be spread across the network, making it difficult to align with the central configuration and maintenance process proposed by the ASOA. Furthermore, instead of deploying multiple CAs and managing other participants' certificates, permissions, and governance files, we envision a more lightweight solution that can be applied even on resource-constrained devices. While the a-priori replication of certificates is a possible optimization technique, their management is still problematic for embedded control units, which the ASOA framework also addresses. Notably, the RTPS implementation [90] deployed on ECUs units does not even support the DDS Security Specification. Finally, incorporating a dedicated security solution tailored for the ASOA framework will facilitate the support for other communication middlewares.

Our efforts focus on a solution that offloads as many security-related operations from ASOA service developers. Furthermore, our solution does not lead to new inter-service dependencies since such would impede the cost-efficient maintenance of the automotive ecosystem. We argue that individual control units do not need a global view of the security architecture but should only be equipped with a minimum of necessary security artifacts. While purely decentralized solutions like the DDS Security Specification require each component to keep track of third-party certificates and a list of global permissions, we adhere to the centralized management approach of the ASOA framework by centrally administering security requirements. Consequently, automotive software components remain decoupled from each other since they only *know* themselves and can be modified, added, or removed at a relatively low cost. Our security process consists of three consecutive phases, as shown in Figure 6.2.

The first one (P1) provides means to associate dataflows of a centrally maintained vehicular architecture with security attributes. For that purpose, we enhance the existing architecture tool. Only at that point, service operators have to become active with regard to security. During the second phase (P2), a trusted platform reads an annotated communication model resulting from P1, converts it into security tokens, and distributes them securely to the ECUs during a one-time initialization step. At last, the third phase (P3) denotes the actual runtime protection of ASOA-based communication. That is, once a service starts running, it uses the previously received tokens to secure outgoing and verify incoming data according to the objectives that have been earlier specified in P1.

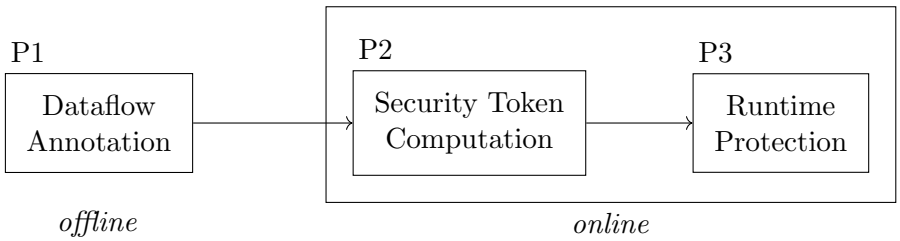

Figure 6.2: Our security process consists of three phases: P1, P2, and P3. In an offline step, service developers annotate dataflows with security attributes in P1. The ASOA Security Platform converts the annotated communication graph into tokens in P2, which are used to protect ASOA traffic during vehicle operation time in P3.

Typically, automotive ecosystems consist of various heterogeneous control units regarding resources and technical capabilities. Although there is a clear tendency for less but more powerful

in-vehicle control units [91], we argue there will still be constrained devices such as sensors or actuators. Also, ECUs may be shielded on an isolated bus due to safety considerations and, therefore, have fewer networking capabilities. We consider those circumstances in P2 and P3 by providing tailored solutions for resource-constrained control units.

The remaining section is structured as follows: We first describe our attacker model and then explain the notation we use in the further course of this work. We also present the benefits of a centralized security solution for automotive use cases and weigh our design decisions against existing work. In the further course, we use the term *service* for ASOA services for simplicity.

6.2.1 Security Goals & Attacker Model

Prominent attacks on road vehicles have demonstrated how automotive systems, particularly those without a human fallback layer, are prone to cyberattacks due to several risk factors, as elaborated in Section 3.1. A widely known risk results from the adversary's ability to manipulate and inject potentially safety-critical traffic, allowing him to control road vehicles (remotely) [18]. Therefore, for a safe driving experience, automotive operators and passengers must rely on the authentic real-time transmission of safety-critical operations, an important finding of the conducted security requirement analysis in Chapter 4.

In contrast, the installation of a fake ECU, e.g., in a dishonest repair shop, makes traffic protection dispensible because such ECU is a legitimate communication node, which, however, acts maliciously. Thus, we require means of verifying and validating the integrity and well-functioning of ECUs before they start action. Another risk factor is compromising security-related infrastructure, such as trust anchors and certificate authorities. Furthermore, modern software frameworks facilitate over-the-air updates, thereby allowing adversaries to infiltrate benign ECU. As a result, the update process must fulfill strong security requirements, such as those outlined in UN Regulation No 156.

We categorize the previously mentioned risk factors into two attacker types: the first describes an adversary connected to the network, i.e., a typical man-in-the-middle attacker conducting spoofing attacks. The second type denotes an adversary residing on an ECU through fake hardware or a compromised update. Currently, the ASOA does neither support authentic nor confidential communication, despite the proven efficacy of the first attacker type in compromising passenger safety. Therefore, this chapter concentrates on that particular type and presents a security solution protecting against the risk of a Dolev-Yao attacker. Our security process thwarts adversaries who can intercept, modify, drop, and inject (ASOA) traffic within the network and impersonate legitimate participants. Meanwhile, we acknowledge that automotive software frameworks eventually have to be capable of dealing with all risk factors. Nevertheless, this chater does not encompass hardware attacks and software manipulation. To guarantee the adversary's absence from control units to prevent illegal code execution, data theft, and software corruption, we refer to Chapter 7.

6.2.2 Design Decisions

The ASOA framework combines elements of both decentralization and centralization. At the network layer, it leverages the DDS standard for decentralized data transmission between services. Simultaneously, the framework employs the centrally functioning orchestrator that maintains a holistic view of all available services and orchestrates their connections based on the desired

operation mode. To stick to the ASOA philosophy of minimizing inter-service dependencies, it is inevitable to store as little security-related information on ECUs as possible. That means a service should not be required to manage security attributes like keys, permissions, or certificates *of others* since this would increase their coupling and complicate the overall system maintenance. We achieve this by introducing another central component, the ASOA Security Platform (ASP).

The ASP utilizes the system specification maintained in the architecture tool to identify legitimate dataflows between services and establish appropriate access policy. Each time a new session starts, the ASP converts these flows into security tokens and disseminates them to the ECUs, enabling secure communication. Like the orchestrator, the ASP possesses a global system perspective, specifically regarding its security specification. That way, we align with the core tenets of the ASOA philosophy, emphasizing decoupled and updatable services while minimizing dependencies. Moreover, this central component coincides with current automotive trends, as evidenced by a German draft law from 2019 [92], which prescribes a central authority for autonomous road vehicles, and recent research on future mobility systems that also advocates for a central component [93]. The flow diagram in Figure 6.3 illustrates the interactions between the ASP, the orchestrator, and ASOA services in an automotive session.

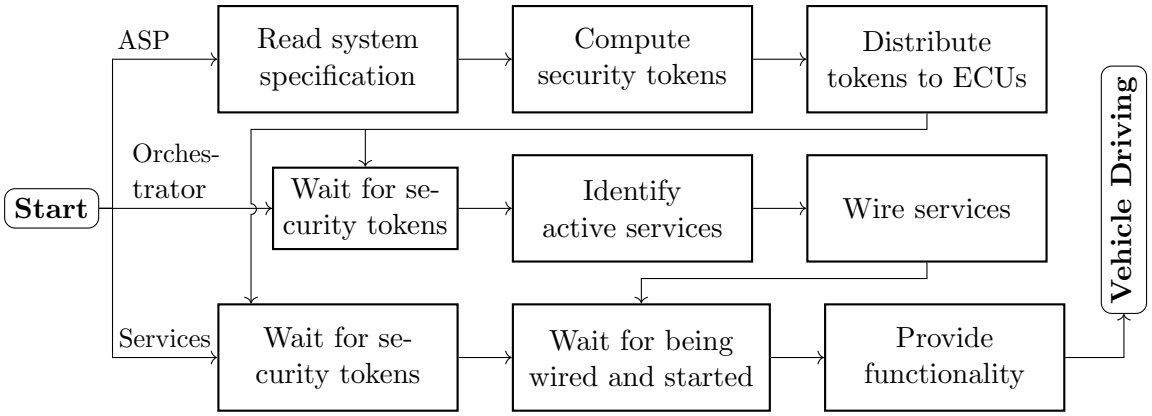

Figure 6.3: At first, the ASP computes security tokens, representing authorized dataflows, by querying the system specification stored in the architecture tool. Upon reception, the orchestrator establishes connections between the services, synergistically enabling the vehicle to navigate.

For simplicity, we treat the ASP as a single component serving as a root of trust for the entire automotive network. This assumption, however, results in a potential single point of failure, as secure service communication becomes unattainable if the ASP is unavailable or even compromised. Possible solutions involve the redundant deployment of the ASP, including fault-tolerant mechanisms to ensure continued operation in the event of failures. By minimizing the attack surface, the probability of successful cyberattacks can be reduced. Hardware security modules offer secure key storage and protection against physical attacks and unauthorized access, further fortifying the root of trust. Additionally, the trust can be distributed across multiple instances of the ASP through techniques such as multi-party computation or a distributed consensus protocol. The secure and highly available design of a computer system like the ASP must be addressed; however, this aspect is considered orthogonal to the objectives of this work. Therefore, we treat the ASP as a *trusted* component in this work, being well aware of the necessity to protect it in a real-life scenario.

6.2.3 Definitions & Notation

The term *automotive ecosystem* describes the network of hardware and software components contributing to fully automatized vehicle driving. It includes ECUs, HPCs, roadside units, control rooms, and cloud servers. The term ECU commonly refers to traditional resource-constrained legacy control units designed for specific automotive functions. In contrast, HPC platforms offer greater computational capabilities and possess a general-purpose nature. Since the ASOA, encompassing our security process, supports both ECUs and HPCs, we employ the term ECU to denote both categories of control units.

Control units communicate through dataflows using the ASOA framework. A specific control unit is *part of* or *involved in* a dataflow if it sends data to or receives data from that particular dataflow. We use the following notation to express our security model formally: While the set G contains all ASOA guarantees of the automotive ecosystem, R describes the set of ASOA requirements, respectively. Recall that guarantees and requirements are communication endpoints of services. Similarly, E is the set of all communication endpoints of the automotive ecosystem, i.e., $\mathsf{E} = \mathsf{R} \cup \mathsf{G}$. An endpoint can never be a guarantee and a requirement simultaneously, which is why $\mathsf{R} \cap \mathsf{G} = \emptyset$ holds. This leads to the set

$$\mathsf{D} = \{(g, r_1, r_2, ..., r_n) \mid g \in \mathsf{G}, r_1, r_2, ..., r_n \in \mathsf{R}\}$$

that contains the legitimate dataflows of the automotive system. We obtain D as a result of the specification and annotation of the communication model. An ASOA dataflow $d \in \mathsf{D}$ consists of exactly one guarantee as a source and an arbitrary number of requirements as sinks. Hence, we express d as an n-tuple, whose first element is the guarantee, followed by its requirements. $\pi_i(t)$ returns the i-th element of a tuple t. According to the definition of D, any guarantee $g_i = \pi_1(d_i), d_i \in \mathsf{D}$ cannot appear as a requirement in any other dataflow d_j, i.e.,

$$\forall d_i, d_j \in \mathsf{D} \mid \pi_1(d_i) \neq \pi_k(d_j), k > 1$$

is invariant on set D. This fact is important since guarantees and requirements will eventually be represented as DDS endpoints on the technical layer. We express the set of requirements as $\mathsf{R} = \{r \in \mathsf{E} \mid \exists d \in \mathsf{D}\ \exists i \in \mathbb{N} \setminus \{0\} : r = \pi_i(d)\}$ and the set of guarantees accordingly, i.e., $\mathsf{G} = \{g \in \mathsf{E} \mid \exists d \in \mathsf{D} \mid \pi_0(d) = g\}$. S describes the collection of ASOA services within the automotive ecosystem. Any service $s \in \mathsf{S}$ consists of a name and a finite number of endpoints, i.e., $\mathsf{S} = \{(\text{name}, e_1, e_2, ..., e_n), e_i \in \mathsf{E}\}$. Be aware that services cannot *share* endpoints since dataflows would not be definite otherwise. This is why

$$\forall s', s \in \mathsf{S} : \pi_i(s) = \pi_j(s') \equiv s = s', i \in [2, |s|], j \in [2, |s'|]$$

holds on S. The relation $\varphi_S(e) : \mathsf{E} \rightarrow \mathsf{S}$ returns the service to which the input endpoint $e \in \mathsf{E}$ belongs. Similar to services, we formally express the control unit as an n-tuple consisting of a name, a cryptographic root identity, and a finite number of services, i.e., $\mathsf{ECU} = \{(\text{name}, \text{root\_identity}, s_1, s_2, ..., s_n) \mid s_i \in \mathsf{S}\}$. Accordingly, $\varphi_E(s) : \mathsf{S} \rightarrow \mathsf{ECU}$ returns the control units on which the input service $s \in \mathsf{S}$ runs. Furthermore, we call P the set of permissions associated with dataflows from vehicle operators. A permission $p_i \in \mathsf{P}$ is a string for which the ASP creates a distinct key to secure the corresponding dataflow. Later, in Section 6.2.5, we explain how to arrange permissions hierarchically to express dependencies between keys and, thus enabling trusted third party (e.g., a domain controller) to derive keys computationally.

6.2.4 Annotating ASOA Dataflows

The first phase of our security process is to allow vehicle operators and service developers to define legitimate dataflows and annotate them with security attributes. In 2018, Kampmann et al. [20] presented an architecture tool for specifying vehicular architectures, which offers a graphical interface for establishing connections between services and defining dataflows. We extend this tool to allow operators to associate dataflows with one of three possible security levels, along with permissions and optionally cryptographic primitives. The security level represents the intended protection for a specific dataflow during runtime. We categorize the security levels as authentication (*AUTH*), encryption and authentication (*AUTH+ENC*), or no protection (*NONE*). Figure 6.4 illustrates how the architecture tool can be utilized to define and annotate two dataflows between four services. The first dataflow will undergo authentication and encryption (l) using the SipHash and the AES algorithm (c). The required key will be derived from the permission "safety" (p). The second dataflow is tagged with the permission "media" and will only be authenticated, again, using the SipHash primitive.

The ASP eventually converts each permission into a distinct key. This allows dataflows of the same security level to be grouped and associated with the same cryptographic key, which we find advantageous for dataflows between the same control units. We pursue a hierarchical key organization scheme, allowing us to group dataflows and delegate the key computation process, if needed, for small embedded ECUs that cannot directly communicate with the ASP (c.f., Section 6.2.5). By default, our security solution assigns a unique permission to each dataflow, but it grants operators the flexibility to optimize key management by defining new permissions.

The collection of annotated dataflows forms a communication matrix stored in a PostgreSQL database. Whenever a new automotive session begins, the ASP parses this database and interprets the dataflows as legitimate. Any other dataflow is considered unauthorized and reported as a security alert during vehicle operation. Technically, the ASP computes a cryptographic key for the permission of each $d_i \in \mathsf{D}$ and distributes it to the control units running the services involved in d_i (c.f., Section 6.2.5).

By annotating dataflows with the mentioned security attributes, vehicle operators only need to specify them once, rather than manually managing security artifacts like certificates, keys, and access control lists and distributing them to individual control units. This simplifies maintenance since changes or updates can be made in one place and propagated to the services through the ASOA Security Platform. In Section 6.4, we give insight into a fully annotated communication graph of a self-driving vehicle and evaluate the automatic processing by the ASP.

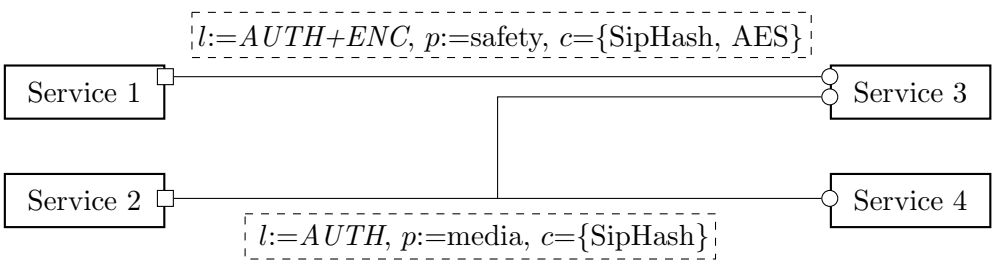

Figure 6.4: A graphical user interface allows vehicle operators to define dataflows between ASOA services and annotate them with a security level, a permission, and cryptographic primitives. The ASOA Security Platform parses the annotated communication matrix and derives individual security tokens from the control units.

6.2.5 Computation and Distribution of Security Tokens

The second phase of our work deals with the computation and distribution of security tokens, enabling ASOA services to communicate securely with each other. Recall that security tokens are derived from the annotated communication model resulting from phase P1. Each ECU sends a key request to the ASP at the beginning of an automotive session. The ASP uses the annotated communication model to look up the legitimate dataflows $\mathsf{D}' \subseteq \mathsf{D}$ of the requesting control unit and creates a security token for all $d \in \mathsf{D}'$. Such a token consists of a cryptographic key, a list of DDS topic names, a security level, and the cryptographic primitives to secure the corresponding dataflow. The ASP uses its secret root key K_{ASP} to compute the key for the permission of a given dataflow. If no permission has been assigned to a dataflow, the ASP creates a unique dummy permission in the background. During the distribution of security tokens, the authenticity of the requesting control unit is implicitly verified to prevent adversaries from intercepting the tokens. Ultimately, the ASP sends a security token for each protected dataflow back to the requesting control unit. Although a potentially large number of services may reside on a control unit, there is precisely one request sent. Hence, the response contains the security tokens of all services of a particular control unit.

Hierarchical Permission Organization

Permissions are associated with dataflows and converted into unique shared keys. The more permissions exist, the more keys are created and distributed. The number of keys would grow fast in large automotive ecosystems without gaining a significant security benefit. Therefore, dataflows between the same control units may share the same key. We achieve this by assigning the same permission to them. Cryptographic keys are renewed in each session. Typically, not every ECU can send a request to the ASP due to technical constraints or, more generally, because they are shielded inside the vehicle. For instance, small sensors or safety-critical control units may not be permitted to communicate with external components. In that case, we argue that a trusted in-vehicle component (e.g., a zone gateway or domain controller) can compute and distribute the keys later. For that purpose, we arrange permissions hierarchically such that they depend on each other. Dataflow permissions are leaf nodes, i.e., they do not have children. In contrast, intermediary permissions are never directly associated with dataflows but instead granted to trusted third-party control units, making them capable of computing the keys of descendant permissions for potentially shielded or technically limited control units.

Formally, we describe our tree-based key computation scheme as follows: While P denotes the set of permissions (c.f., Section 6.2.3), we call the collection of cryptographic keys $\mathsf{K} = \{k_1, k_2, ..., k_n\}$. The permissions are arranged in a directed rooted graph $\mathcal{T}$, allowing us to describe dependencies between them. We consider two permissions p_i and p_j dependent if p_i is an ancestor of p_j. The root permission p_0 must only be known to the ASP since all other permissions depend on p_0. Hence, the owner of p_0 can compute all keys $k_i \in \mathsf{K}$. Figure 6.5 shows an example permission tree. Each permission is assigned a unique index, growing from left to right and indicating its position within the tree. This index is used to produce a unique key for the corresponding permission. We call $Idx(:)\mathsf{P} \rightarrow \mathbb{N}$ the function that returns that index of a given permission. For instance, $Idx(Engine)$ returns 5 with respect to the example tree given in Figure 6.5. A component can only compute the key $k_i \in \mathsf{K}$ for permission p_i if it has an ancestor p_i. Hence, the hierarchically organized permission tree $\mathcal{T}$ leads to dependencies between keys. That means two keys depend on each other if their permissions are also dependent. For example, suppose the ASP grants the permission p_3= “Sensors” to a trusted in-vehicle sensor processing unit. In that case, the latter

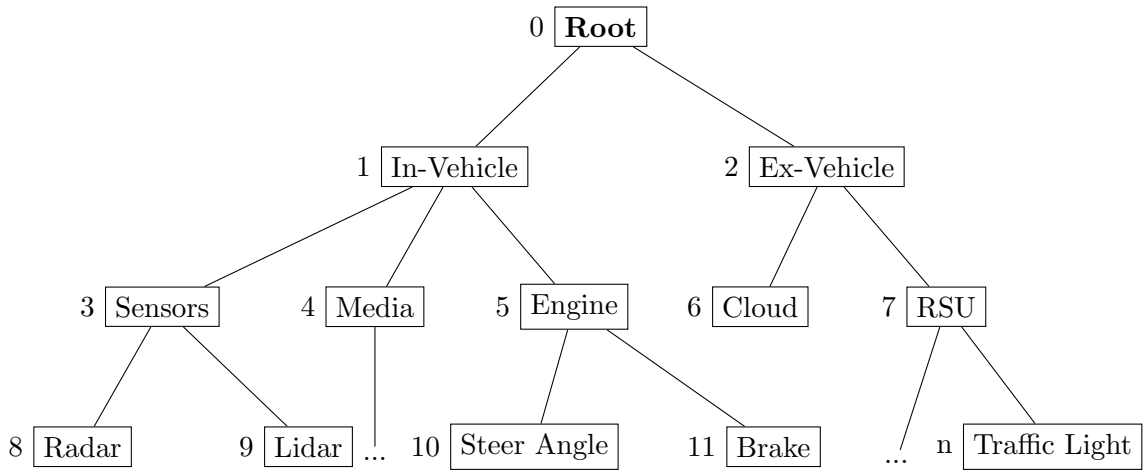

Figure 6.5: Permissions are organized hierarchically. They are numbered from left to right. If an intermediate permission is granted, all child permissions are also granted. This way, trusted domain or zonal controllers can derive cryptographic keys and pass them to shielded control units.

can then compute the keys k_8 and k_9 for p_8= "Radar" and p_9= "Lidar". Eventually, k_8 and k_9 are forwarded to the target control units that cannot directly communicate with the ASP.

Key Computation

Technically, we realize the hierarchic key computation scheme by recursively applying a secure hash function *Hash* on the root secret k_0 associated with the root permission p_0. Consequently, a permission enables its owner to compute the keys of its descendants but does not reveal anything about its ancestors or neighbors. It is generally impossible to compute an ancestor permission as this requires finding pre-images of *Hash*, which would violate the cryptographic properties of hash functions. Thereby, the permission index indicates how often *Hash* is applied. Our approach is similar to keyed hashed chains [94], a well-studied and established way of deriving cryptographic keys from a single input source. However, we do not build a flat hash chain but consider a tree structure instead. In the following, we use the function $Prnt : \mathsf{P} \rightarrow \mathsf{P}$ to query the parent node of a permission. Moreover, the function $CalcKey : \mathsf{P} \rightarrow \mathsf{K}$ calculates the key k_i for the permission p_i and is defined as follows:

$$CalcKey(p_i) = \begin{cases} \mathsf{KDF}\left(K_{ASP} \mid\mid K_{\text{session}},\ slt\right) & i = 0 \\ \mathsf{HMAC}^{Idx(p_i)}\left(slt,\ \ CalcKey(Prnt(p_i))\right) & \text{else} \end{cases}$$

We distinguish between the computation of k_0 and k_i with $i > 0$. In the former case, we first concatenate the root secret K_{ASP} with a randomly chosen session key K_{session}. Recall that K_{ASP} is the root secret of ASP and does not change over time. Subsequently, we feed the result into a Key Derivation Function (KDF) to obtain k_0.

As an example, the Password-Based Key Derivation Function 2 (PBKDF2) [95], or the Scrypt [96] could serve as KDF. While PBKDF2 protects against dictionary and rainbow table attacks, it is not resistant to parallelization attacks. In contrast, Scrypt requires both computational and memory resources. We leave the ultimate choice of the KDF to the vehicle operator. When it comes to the computation of key k_i with $i > 0$, we repeatedly compute an HMAC of the parent key, i.e., of the key of permission $Prnt(p_i)$. More precisely, we call the HMAC function as often as the tree index of p_i indicates. In that way, we ensure that every permission receives a unique

key and simultaneously map the tree hierarchy on the derived keys. Since the computation of a particular key requires the parent key, an intermediate permission enables its owner to derive the keys of all child permissions computationally.

Once the ASP receives a key request from a particular control unit $\mathcal{E}_\text{i} \in \mathsf{ECU}$, it conducts the steps shown in Algorithm 1. That means it first queries the dataflows $\mathsf{D}' \subseteq \mathsf{D}$ that involve at least one service running on $\mathcal{E}_\text{i}$. We express D' as $\mathsf{D}' = \{d \in \mathsf{D} \mid \exists i \in \{0, ..., |d|\} \mid \varphi_E(\varphi_S(\pi_i(d))) = \mathcal{E}_\text{i}\}$. The ASP reads the permission of each $d_i \in \mathsf{D}'$ and computes the key using the above-mentioned *CalcKey* function. Moreover, it queries the desired security level and retrieves the cryptographic primitives to be used. After that, the ASP determines the topic names of the DDS streams that emerge from each $d \in \mathsf{D}'$. This step is necessary because ASOA dataflows are internally converted into DDS streams that can be identified by the topic name. As Section 6.1 explains, an ASOA dataflow can internally lead to more than one DDS stream since the ASOA framework carries quality and metadata on separate streams. The name of an ASOA guarantee is used to determine the topic name of the resulting DDS stream. For that purpose, ASOA framework offers an encoding scheme that inputs a guarantee and the resulting topic names. The ASP constructs a security token for each $d_i \in \mathsf{D}$ consisting of the key, the DDS topic names, the security level, and the cryptographic primitives to be used. Later, our runtime protection unit unpacks the token and attaches the key, the security level, and the cryptographic primitives to the corresponding DDS streams of dataflow d_i. Using the following scheme, the ASP transfers the security tokens to the requesting control unit.

Algorithm 1 *GetSecTokens*($id_{\mathcal{E}_\text{i}}$)

```
1: tokens ← empty()
2: D′ ← GetDataFlows(id_Eᵢ)
3: for d_i in D′ do
4:     sl_j ← d_i.GetSecurityLevel()                ▷ defined in P1
5:     pr_j ← d_i.GetCryptPrimitives()              ▷ defined in P1
6:     p_j ← d_i.GetPermission()                    ▷ defined in P1
7:     k_j ← CalcKey(p_j)                           ▷ see Section 6.2.5
8:     tn_j ← d_i.GetTopicNames()
9:     token_j ← (k_j, tn_j, pr_j, sl_j)
10:    tokens.Add(token_j)
11: end for
12: return tokens
```

Token Exchange Protocol

The construction and transmission of security tokens are embedded into an exchange protocol executed between ECUs and the ASP when a new automotive session begins. The scheme consists of three main messages. At first, a control unit $\mathcal{E}_\text{i}$ sends a request to the ASP before launching its services. Next, the ASP replies with encrypted security tokens for $\mathcal{E}_\text{i}$ and implicitly verifies the identity of $\mathcal{E}_\text{i}$ to prevent man-in-the-middle attacks. Finally, $\mathcal{E}_\text{i}$ acknowledges the successful execution of the protocol and provides the received tokens to our runtime protection unit.

The security tokens are encrypted with a shared identity key during transmission. The encryption protects them from eavesdropping and serves as implicit authenticity proof since the identity key is only known to the requesting $\mathcal{E}_\text{i}$ and the ASP. We deliberately use an identity key and abstain from certificates because control units do not need to provide an identity to third parties but

only negotiate with the ASP. As the identity key persists between automotive sessions, storing and protecting it securely from illegal access and leakage is crucial. While we assume secure, non-volatile memory on larger control units, resource-constrained devices do not necessarily provide hardware support. Therefore, we present two variations of our exchange protocol, which differ in how secure identity information is obtained from a control unit. The first version (TokDist-SecMem) reads an identity key from secure memory and, thus, is specifically applicable to resourceful control units. In contrast, the second one (TokDist-PUF) exploits unique physical memory characteristics using a PUF (cf. Section 2.6) to prove its identity to the ASP.

TokDist-SecMem The requesting control unit $\mathcal{E}_\text{i}$ is assumed to possess a unique identity key $K_{\mathcal{E}_\text{i}}$ that is shared with the ASP and stored in secure memory. This key has been agreed on in a one-time initialization step (e.g., during fabrication), together with a multiplicative group of integers modulo p and a primitive root g. As illustrated in Figure 6.6, three messages are exchanged in total. At first, $\mathcal{E}_\text{i}$ selects a private key $sk_{\mathcal{E}_\text{i}}$ from $\mathbb{Z}_p$ and computes the corresponding public key $pk_{\mathcal{E}_\text{i}} = g^{sk_{\mathcal{E}_\text{i}}} \mod p$. Then, $\mathcal{E}_\text{i}$ creates a key request m, consisting of its identifier $id_{\mathcal{E}_\text{i}}$ and the public key $pk_{\mathcal{E}_\text{i}}$. Next, $\mathcal{E}_\text{i}$ computes the MAC mac on m using the identity key $K_{\mathcal{E}_\text{i}}$ and sends both m and mac to the ASP. Upon reception, the ASP checks the authenticity of m by verifying mac and computes the security tokens $sectok$ for $\mathcal{E}_\text{i}$ using Algorithm 1. Similar to $\mathcal{E}_\text{i}$, the ASP selects a private key sk_{ASP}, generates its public key pk_{ASP}, and computes $K'_{\mathcal{E}_\text{i}} = pk_{\mathcal{E}_\text{i}}^{sk_{ASP}} \mod p$, i.e., it performs a Diffie-Hellman key exchange.

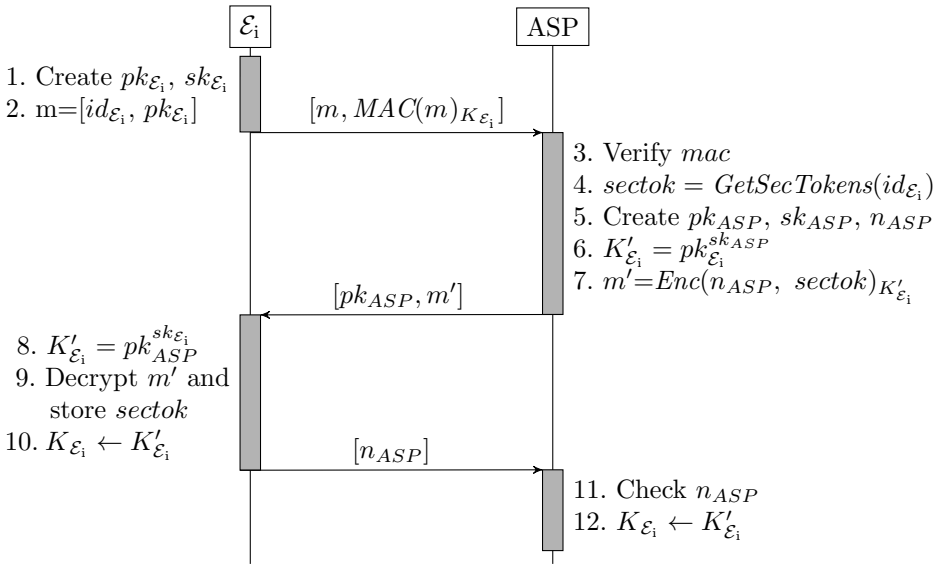

Figure 6.6: ECU $\mathcal{E}_\text{i}$ with secure memory sends a request (2) to the ASOA Security Platform (ASP) containing its ID and a Diffie-Hellman public key. The ASP verifies the request (3), creates (4) and encrypts (7) the security tokens using a shared DH key (5,6), and eventually sends it back to $\mathcal{E}_\text{i}$ The requesting control unit decrypts (9) the tokens and updates its identity key (8,10).

After having done so, the ASP selects a random nonce n_{ASP} and encrypts it together with $sectok$ using $K'_{\mathcal{E}_\text{i}}$. Eventually, it sends the ciphertext and pk_{ASP} back to $\mathcal{E}_\text{i}$. The encryption of $sectok$ prevents its disclosure while being transmitted to $\mathcal{E}_\text{i}$ and likewise serves as an implicit authenticity proof of $\mathcal{E}_\text{i}$ to the ASP since only $\mathcal{E}_\text{i}$ is capable of computing $K'_{\mathcal{E}_\text{i}}$, required to decrypt the data. For that purpose, $\mathcal{E}_\text{i}$ multiplies its private key $sk_{\mathcal{E}_\text{i}}$ with ASP's public key and obtains $K'_{\mathcal{E}_\text{i}}$. At that point, $\mathcal{E}_\text{i}$ uses $K'_{\mathcal{E}_\text{i}}$ to decrypt the nonce n_{ASP} as well as $sectok$. It acknowledges the successful reception of the security tokens by sending n_{ASP} to ASP. If the ASP successfully verifies n_{ASP}, it updates $\mathcal{E}_\text{i}$'s identity key and stores it in secure memory to protect it from illegal

access. Without such an update mechanism, the $K_{\mathcal{E}_\text{i}}$ would persist throughout the vehicle's lifetime, which may be decades. However, a long-living key is more likely to be disclosed or leaked because the attacker gains more computational power over time.

TokDist-PUF Not all ECUs possess secure memory for storing security tokens. To address this issue, we propose utilizing unique physical memory properties as an alternative method to obtain an identity key. Specifically, we employ a PUF to extract an unforgeable and distinct fingerprint from a resource-constrained control unit. This fingerprint undergoes several processing steps to derive a cryptographic key, which can be used for the secure transmission of the security tokens. A PUF takes advantage of the deterministic exhibition of unique physical properties, which are typically random and unpredictable, such as minor variations in transistor doping. Memory itself is often an intrinsic PUF due to its distinctive access behavior. For instance, an SRAM PUF uses the values of SRAM memory cells after bootup as a fingerprint [97]. In contrast, row hammering [98] and latency [99] PUFs rely on the decay of DRAM cells to create a fingerprint. Ideally, a PUF produces the same output (response) for the same input (challenge). In this context, a challenge refers to a specific memory region whose physical properties are measured and encapsulated in a fingerprint. The PUF response is fed in a fuzzy extractor, a vital post-processing step to remove noise and to receive a stable output. Thus, a PUF enables us to derive an identity proof from memory characteristics without explicitly storing a key, as we did previously. Consequently, we can deploy our security solution even on embedded devices lacking secure, non-volatile memory. During a one-time initialization step, a finite number of challenges $\mathcal{C}_{\mathcal{E}_\text{i}}$ is created and provided into the PUF of $\mathcal{E}_\text{i}$. As a result, we obtain a set of responses $\mathcal{R}_{\mathcal{E}_\text{i}}$. Subsequently, we define $\phi : \mathcal{C} \rightarrow \mathcal{R}$ as the function that produces the response for a given challenge. Figure 6.7 illustrates the process wherein three messages are required to securely transmit the tokens from $\mathcal{E}_\text{i}$ to ASP.

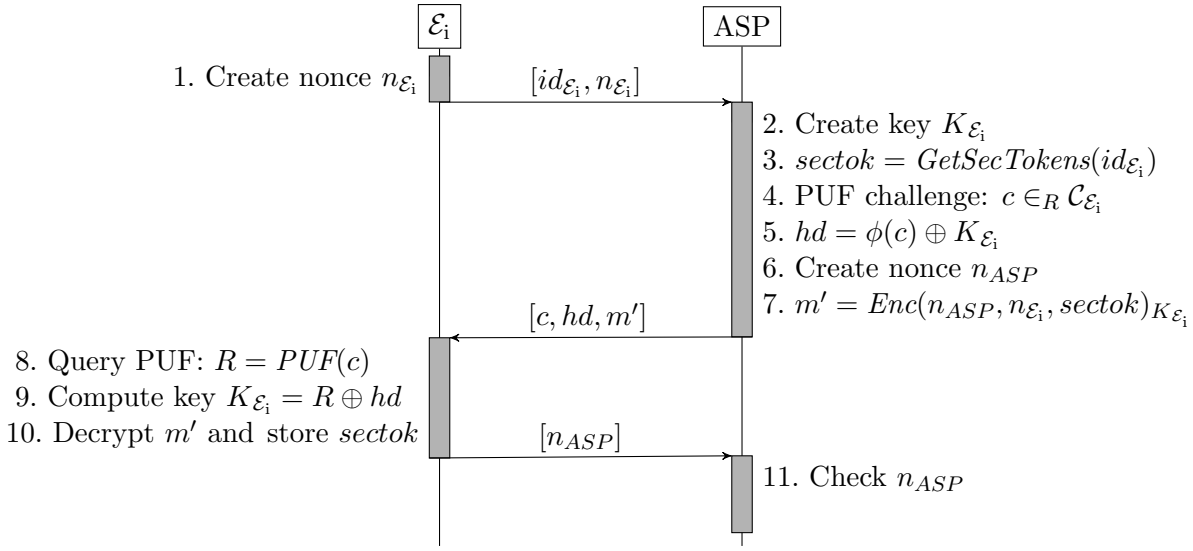

Figure 6.7: ECU $\mathcal{E}_\text{i}$ without secure memory uses a PUF as identity proof. It first sends a random nonce and its ID to the ASP (1), which generates a secret key (2) $K_{\mathcal{E}_\text{i}}$, creates the security tokens (3), randomly selects a PUF challenge (4) and encrypts the tokens using the expected PUF response (7). Upon reception, $\mathcal{E}_\text{i}$ queries its PUF (8), computes the key (9), and decrypts the security tokens (10).

At first, $\mathcal{E}_\text{i}$ sends its identifier $id_{\mathcal{E}_\text{i}}$, along with a randomly selected nonce $n_{\mathcal{E}_\text{i}}$ to the ASP. The latter computes the security tokens *sectok* as explained in Algorithm 1. Furthermore, it creates a key $K_{\mathcal{E}_\text{i}}$ and uses it to encrypt another random nonce n_{ASP} and *sectok* . In addition, the

ASP randomly picks a challenge $c \in \mathcal{C}_{\mathcal{E}_\text{i}}$ and queries the corresponding response $r = \phi(c)$. Then, it builds the helper data hd that uses r as a one-time pad to disguise $K_{\mathcal{E}_\text{i}}$, i.e., $hd = r \oplus K_{\mathcal{E}_\text{i}}$. Eventually, the ASP sends the reply message $[c, hd, Enc(n_{ASP}, n_{\mathcal{E}_\text{i}}, sectok)_{K_{\mathcal{E}_\text{i}}}]$ to $\mathcal{E}_\text{i}$.

Upon receiving the reply message, $\mathcal{E}_\text{i}$ first checks $n_{\mathcal{E}_\text{i}}$, then employs c to challenge its PUF and eventually obtains the response r'. If $\mathcal{E}_\text{i}$ is authentic, then $r' = r$ holds. $\mathcal{E}_\text{i}$ computes $K_{\mathcal{E}_\text{i}} = r' \oplus hd$, thereby reconstructing $K_{\mathcal{E}_\text{i}}$. Note that other control units can neither reproduce r' nor decrypt *sectok* since the PUF response r' is tied to unique physical properties of $\mathcal{E}_\text{i}$. $\mathcal{E}_\text{i}$ decrypts *sectok* and n_{ASP} and sends the latter back to ASP. By doing so, $\mathcal{E}_\text{i}$ acknowledges the protocol's successful termination and proves its authenticity to the ASP.

Finally, $\mathcal{E}_\text{i}$ launches its services after providing them the security tokens through our runtime protection unit.

6.2.6 Runtime Protection Unit

The Runtime Protection Unit (RPU) provides essential functionalities for securing and verifying dataflows during vehicle operation. Its primary purpose is to enforce the security objectives specified in P1 by utilizing security tokens obtained from the ASP. Upon reception, the control unit stores these tokens in a shared memory segment, ensuring accessibility to all services on the respective machine. Next, a launching service converts its guarantees and requirements into DDS endpoints and then utilizes the RPU to associate the security tokens with those endpoints. This step is crucial for later matching data from endpoints to ASOA dataflows. Technically, we achieve this association by modifying the eProsima RTPS [100] as well as the embeddedRTPS [90] implementation such that the tokens' content, like the security level, the cryptographic key, and the primitives are attached to the corresponding endpoints. Ultimately, this enables the RPU to effectively associate frames with their original ASOA dataflow and enforce the earlier defined security objectives.

The RPU comprises two essential components: the encryption engine and the authentication engine, both accessing the crypto core, a library providing cryptographic primitives. The crypto core allows for easy extension through system updates to introduce better and more secure primitives over the vehicle's lifetime. The encryption engine is situated on the edge of the ASOA

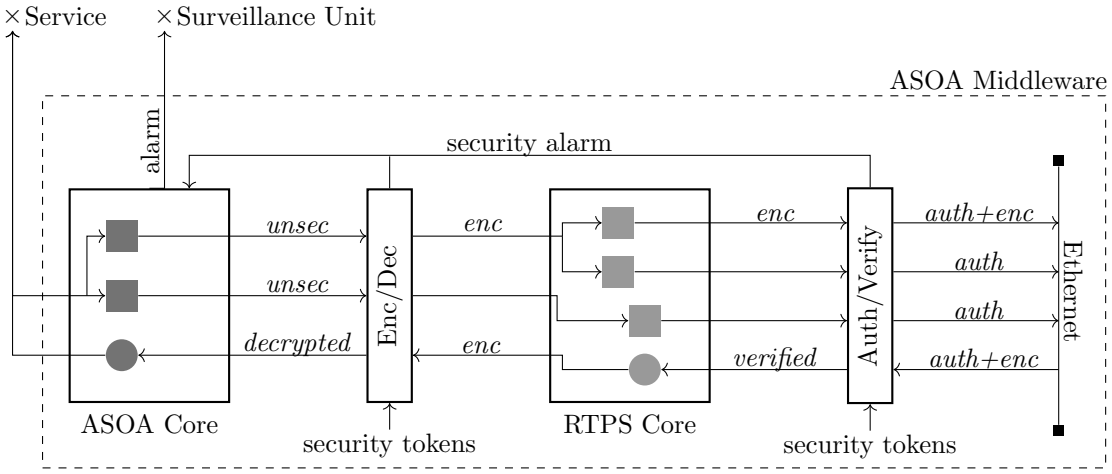

Figure 6.8: The encryption engine produces ciphers of the payload data at the edge of the ASOA core. In contrast, the authentication engine intercepts RTPS frames and guarantees and verifies their authenticity. In the event of a failed cryptographic operation, a security alarm is reported to a central surveillance unit.

layer. In contrast, the authentication engine resides between the RTPS layer and the Ethernet interface, as depicted in Figure 6.8. In scenarios where dataflows do not require protection, both engines forward the data. In contrast, if the security level prescribes authenticity and confidentiality, the encryption engine produces a cipher of the payload. This cipher is then passed to the RTPS Core, transforming it into RTPS frames. Subsequently, the authentication engine intercepts these frames, computes an authentication tag, and ensures data freshness by maintaining a unique counter for each endpoint. The authentication tag provides evidence of the ciphertext's authenticity and the entire RTPS frame, including the header and all submessages. This approach prevents tampering at the RTPS layer while safeguarding confidential content. Thus, our approach follows an authenticated encryption paradigm by encrypting the data to be transmitted and subsequently authenticating both the ciphertext and the remaining RTPS frame.

The distinct layers at which the encryption and authentication engine operate in our software stack make the direct application of an Authenticated Encryption with Associated Data (AEAD) solution infeasible. AEAD algorithms combine encryption and authentication into a single operation, providing benefits in terms of efficiency and security. A prominent example is the combination of the stand-alone algorithms ChaCha20 [101] and Poly1305 [102], both part of our crypto core, into an AEAD algorithm [103]. Our deliberate decision to deploy the encryption engine at the ASOA core's edge and the authentication engine at the RTPS layer ensures authenticity for payload data, header fields, and meta RTPS frames that are invisible on the ASOA layer. Most prominently, RTPS utilizes discovery, heartbeat, and acknowledgments to discover the network topology and to confirm earlier received messages. Without security precautions, an attacker could easily introduce non-existent control units by injecting false discovery messages into the network. By leaving the encryption engine on the ASOA layer, the RPU is not fully coupled to a specific RTPS implementation, which would complicate future support for other communication middlewares. Another aspect of why we chose to encrypt data on ASOA layer is the evitable computational overhead that would occur when encrypting data on RTPS layer *after* the frame has already been created. In such cases, the ciphertext may be longer than the original message, requiring a new frame to be produced.

The RPU plays a dual role in the system, ensuring the security of outgoing messages and verifying the integrity and authenticity of incoming traffic. Once messages are successfully verified, they undergo processing by the RTPS core and are assembled into a dataflow within the ASOA Core. Eventually, they are made accessible to services through the corresponding requirement. As a result, the RPU remains concealed from the services and their developers, operating transparently in the background.

In the event of failed verification or decryption, the associated engine raises a security alarm, signaling a potential security breach. These alarms are intended to be reported to a central surveillance unit. We have established an interface to facilitate the reporting of security alarms, enabling immediate assessment and potential response. We argue that the ability to interpret security-related incidents is of utmost importance. For instance, the failure to verify the authenticity of a data frame could be attributed to a transmission error or, in more severe cases, an attack. Therefore, equipping road vehicles and the broader automotive ecosystem with dedicated protection and defense mechanisms and the capability to comprehend potential warnings and alerts is crucial to ensuring overall security. We will present and discuss a possible solution to reacting to security incidents in Chapter 8. Our security process contributes to this demand by

facilitating the centralized collection of security alarms, which can be analyzed and interpreted to enhance situational awareness and response capabilities.

In summary, the RPU realizes dataflow protection during runtime, using the security tokens created and supplied by the ASP every time a new session starts. This approach ensures that changes made to the security model are securely propagated from the architecture design tool to the network layer of each control unit, eliminating the need for manual intervention. As a result, service developers do not need to worry about improper or missing security initialization as long as the legitimate dataflows have been defined in P1. Hence, we claim that our solution adheres to the principles of the ASOA philosophy, preserving a high degree of cost-efficient maintainability.

6.3 Formal Verification

This section evaluates the security of the proposed token distribution protocol. We employ formal verification methods to ensure the protocol can protect tokens from manipulation during transmission. Formal verification is a rigorous process that involves proving or disproving whether a system satisfies specific properties. At first, we represent both protocol variants as formal models and then utilize the Tamarin model checker [50] to prove their correctness. Note that our security analysis focuses solely on the token distribution protocol. It does not encompass the cryptographic algorithms employed by the RPU, as they have been extensively evaluated through cryptanalysis in prior works.

6.3.1 Tamarin

The Tamarin prover is an open-source tool designed for protocol verification, employing multiset rewriting rules to model the behavior of a given protocol. In this context, a multiset represents the current state of the protocol, with the initial state being an empty multiset. Tamarin uses rules consisting of a premise, a conclusion, and optional action facts to model state transitions. Generally, a rule is executable if the current state includes all the facts specified in its premise. Moreover, Tamarin executes rules concurrently in potentially various instances. The resulting trace incorporates the action facts, while the state facts from the premise are replaced by those derived from the rule's conclusion. Tamarin uses first-order logic formulas over traces of action facts and time points to express security properties in lemmas. During the verification process, Tamarin searches for traces to confirm or discard the assumed security property expressed in the lemma. Tamarin assumes the same attacker capabilities as our attacker model presented in Section 6.2.1. That means a Dolev-Yao adversary controls the untrusted network, being capable of manipulating, injecting, delaying, replaying, and dropping messages arbitrarily often.

6.3.2 Protocol Specification

The protocol variants, as depicted in Figures 6.6 and 6.7, consist of four distinct steps. Each step is modeled as a Tamarin rule, ensuring a formal representation of the protocol's behavior. Listing 6.1 expresses the initial step where an ECU initiates a token request. The rule's premise encompasses three crucial facts: the generation of a fresh nonce (`ecunonce`), the utilization of a shared identity key (`id_key`), and the involvement of a temporal secret key (`sk`). In contrast, the conclusion contains the resultant token request (`req`) and its corresponding MAC. Apart from the four protocol steps, the additional *Deploy_ID_Keys* rule is introduced to deploy the identity key on each ECU. We model the security tokens as a random number that the ASP

creates every time it receives a request. For those interested in the complete proof, please refer to annex B.1.

```
rule ecu_send_token_request:
 let
  ecupubkey = 'g'^sk
  req = <$ECU, ecupubkey>
 in
 [!ID_Key_ECU(id_key), !SK($ECU, sk)]
 --[ Send($ECU, req), Secret(id_key), Secret(sk) ]->
 [Out(<req, mac(req, id_key)>)]
```

Listing 6.1: This Tamarin rule models an ECU requesting security tokens according to the proposed token distribution protocol depicted in Figure 6.6.

6.3.3 Security Properties

Next, we formulate lemmas to establish and validate each desired security property. This step is crucial as the absence of a lemma could obscure a protocol vulnerability despite yielding seemingly valid proof. Our analysis encompasses five lemmas, each addressing a specific aspect of the token distribution protocol. Two lemmas explicitly assert that both the ECU and the ASP possess identical security tokens and an identical updated shared identity key after executing either protocol variant. An additional lemma emphasizes that the tokens and identity key must never be exposed to the adversary. While the inherent characteristics of a PUF guarantee source authenticity within TokDist-PUF, the first protocol variant necessitates the utilization of a MAC. As cryptographic primitives such as MACs are also symbols in Tamarin, we require more lemmas to express their properties. Two further lemmas are formulated to ensure a token request's authentic and fresh transmission from the ECU to the ASP. Failure to uphold these properties would enable an adversary to forge a token request, thereby gaining the ability to conceal the updated identity key and eventually gain unauthorized access to the security tokens. Besides, we introduce two operational lemmas to ensure the termination of the token distribution protocol and validate the successful updating of the identity keys on both the ECU and the ASP.

6.3.4 Intermediate Conclusion

We successfully verified the lemmas using the Tamarin model checker, confirming that our protocol satisfies the properties of ensuring the confidential, authentic, and fresh transmission of security tokens. Hence, our solution effectively protects against adversaries according to our attacker model, including scenarios such as installing a fake ECU. In such cases, the attacker's capabilities would need to surpass those assumed by our attacker model, as we proved that the secret identity key is never revealed on the network.

6.4 Evaluation

This chapter evaluates our security process on a fully automatized prototype vehicle. Since the ASOA framework has been designed for real-time automotive systems, we especially focus on time behavior. At first, we present the evaluation platform, a self-driving road vehicle, in Section 6.4.1 and then inspect the underlying communication model by analyzing the security annotations in Section 6.4.2. Afterward, we perform a thorough runtime analysis in Section 6.4.3. That means we first investigate the timing behavior of the Crypto Core on various control units and identify

the fastest-performing ones. Then, we measure the overall time overhead that our RPU incurs while the vehicle is in motion. Finally, we implement a Physical Unclonable Function on an Field Programmable Gate Array (FPGA) in Section 6.4.4 and demonstrate how it is capable of authenticating embedded, potentially resource-constrained control units during the proposed token exchange protocol.

6.4.1 Evaluation Platform

We evaluate our security process in the reference vehicles presented in Section 4.1. Recall that these vehicles are built on an innovative bionic E/E architecture inspired by the human nervous system. Figure 4.1 visualizes the vehicles' hardware architecture. Across the main event chain, environment perception, route and trajectory planning, control system, and wheel actuation occur. The primary event chain involves environment perception, route and trajectory planning, control system, and wheel actuation, all implemented across four main zones containing four sensor modules, the cerebrum, the brainstem, and four dynamic modules.

Additional ECUs carry out tasks such as battery management, door control, localization, and the HMI. The hardware deployed in the vehicle encompasses various types, including high-performing computing systems, small embedded devices, real-time processing units, and off-the-shelf components, as illustrated in Table 4.1.

6.4.2 Dataflow Analysis

After enhancing the architecture design tool for vehicular architectures, we engaged a team of researchers to annotate the dataflows of the reference system using the graphical user interface provided by this tool. The latter facilitates the storage of the vehicle architecture, encompassing service contracts, interfaces, service dataflows, and security annotations, in a PostgreSQL database. Subsequently, we developed a server that connects to this database, dynamically parses the stored information, and exposes the annotated communication model to the ASP through a REST interface. Furthermore, we have developed a website[1] where service developers and vehicle operators can manually examine the dataset. In our analysis, we examined the communication graph of the reference vehicle, which is represented in XML format, and extracted relevant statistical data. The findings indicate that, on average, a control unit executes 2.77 services and possesses 14.77 endpoints. The system comprises 26 active control units and 61 services, with communication occurring through 113 dataflows. Within this context, we established 113 guarantees associated with 269 requirements, considering the possibility of requirements appearing in multiple dataflows. Depending on the vehicle operation mode, 48 requirements receive data from two different guarantees, 32 are wired to four guarantees, and 15 receive data from up to five guarantees. This observation implies that dynamic rescheduling of dataflows is prevalent within the vehicle, showcasing the flexible administration of software components, which is one of the key advantages of the ASOA framework. In contrast, 174 requirements are consistently connected to the same guarantee, indicating static data transmission. Interestingly, there are eight dataflows where the requirements and guarantees reside within the same control unit. Security tokens are not generated in such cases since no data is transmitted over the network. Instead, the data exchange occurs within these guarantees and requirements through shared memory, thus falling outside the purview of our attacker model.

[1] `https://asoasecurity.seceng.fim.uni-passau.de/`

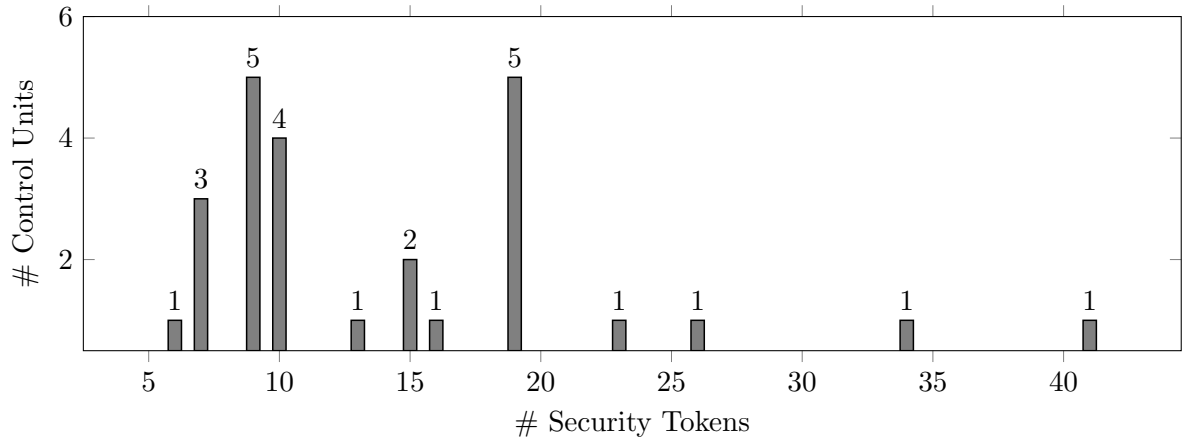

Figure 6.9: ECUs require 15 security tokens on average. The brainstem and the vehicle gateway are outliers, with 43 and 34 tokens, respectively. In contrast, the battery management needs only six tokens.

We generate a security token for each dataflow for the remaining control units, as explained in Section 6.2.5. Figure 6.9 illustrates the number of control units receiving a particular number of tokens. On average, the security platform transmits 15 tokens to each control unit. Since a single token occupies 49 bytes, $15 \cdot 49 = 735$ bytes are necessary on average to store all tokens. It is important to note that this memory usage poses no issues for any deployed ECUs, including the small embedded ones. Thus, we consider the imposed memory overhead to be negligible. In comparison, X.509 v3 certificates, as required by the DDS Security Specification, typically range in size from 1 KB to 4 KB. As discussed in Section 2.5, the DDS Security Specification mandates participants to store and collect various certificates, signed permission files, and governance documents. While this may not pose a problem for powerful control units, it can be challenging for resource-constrained devices with limited memory. Therefore, we find our solution is better suited for embedded devices.

Our analysis observed that the brainstem and the vehicle gateway are outliers, requiring the largest number of security tokens (41 and 34, respectively). This observation aligns with the vehicle architecture, as the brainstem lies on the main event chain between two major zones, the cerebrum and the spinal cord. In addition to examining the set of legitimate dataflows, we also evaluated the manually crafted permission tree, which differentiates between safety-critical and non-safety-critical, as well as in-vehicle and ex-vehicle dataflows. Currently, the permission tree consists of 62 permissions for 113 dataflows. Hence, the ASP generates 62 cryptographic keys for each session, a reasonable number for an automotive system. In contrast, if the DDS Security Specification were employed, the number of cryptographic keys generated would be significantly higher. Specifically, at least 226 cryptographic keys would be required due to the 113 ASOA dataflows necessitating twice as many DDS topics to accommodate the transmission of quality data on separate topics, as discussed in Section 2.5. It is important to note that this figure represents a conservative estimate, as distinct cryptographic keys are computed for each data link in DDS security. Therefore, considering that many ASOA dataflows consist of multiple requirements (i.e., DDS subscriptions), the actual number of cryptographic keys would be even higher.

6.4.3 Runtime Analysis

We now focus on the performance of the RPU. In particular, we evaluate the timing behavior and identify the overhead during vehicle runtime.

Evaluation Sizes

The data size processed significantly impacts the runtime performance of cryptographic primitives and the transmission through the network. Therefore, we first identify reasonable packet sizes for meaningful runtime analysis and then use them for subsequent evaluation steps. For that purpose, we recorded ASOA-based traffic using the Wireshark[2] tool and inspected the sizes of the captured packets. Table 6.1 illustrates the relative frequency of a specific packet length. Obviously, there are two peaks: 51% of the captured packets lie between 40 and 159 bytes,

Packet Length (Bytes)	Average Packet Size (Bytes)	Percent
0-39	20	0%
40-159	**100**	**51%**
160-639	400	2%
640-1279	**960**	**7%**
1280-2559	**1920**	**40%**
2560-5119	3840	1%

Table 6.1: Average length of ASOA packets

while another 47% have a size of 640 to 2559 bytes. A detailed inspection reveals that metadata (heartbeats, acknowledgments, DDS discovery requests, ...) causes the first peak, while the second represents the ASOA service payload, i.e., the automotive data being moved along the event chain in the road vehicle. It may seem confusing that transmitting large amounts of data (e.g., sensor data) does not lead to larger frames on the wire. We attribute this fact to the internal fragmentation and optimization by the ASOA cores. For the following evaluation experiments, we use representative test packets of the average size of the identified peaks (i.e., 100 and 1440 bytes).

Crypto Core

The crypto core (c.f., Section 6.2.6) provides implementations of various cryptographic primitives used for securing ASOA communication during vehicle operation time. It currently covers the categories *authentication*, *encryption*, and additionally provides two *elliptic curves* (Curve25519 [104] and FourQ [105]) for efficient computations. Besides, the crypto core consists of seven authentication (Blake2, Blake3, Chaskey, HMAC-SHA256, Poly1305AES, SipHash, Skein) and four encryption (AES-ECB, Chacha20, Speck64/128, Sparx64/128) algorithms. Our goal is to keep the overall runtime impact of our security solution as low as possible. Therefore, this evaluation experiment serves to identify the *fastest* algorithms that are afterward brought into the reference vehicle. Since all of the mentioned cryptographic primitives are well-established and have been reviewed by experts, we abstain from a security analysis. Instead, we investigate how they perform on *five* representative control unit architectures and determine the best-performing primitive for each category mentioned above. By considering the evaluation results from different architectures, we try to compensate for potential hardware-specific optimizations that might

[2] `https://www.wireshark.org/`

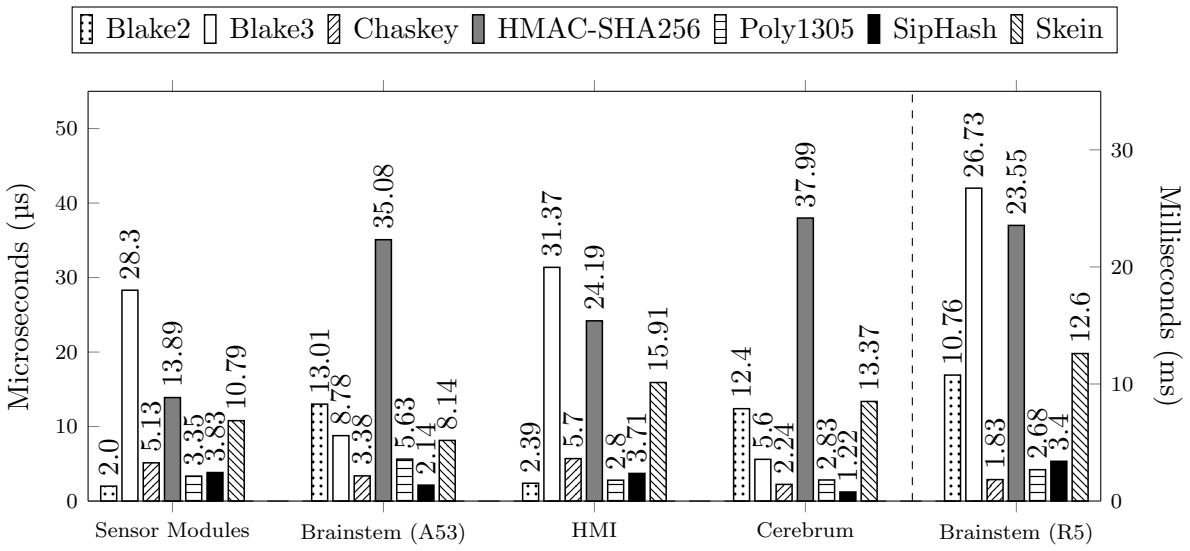

Figure 6.10: As part of evaluating the crypto core, we measured the time consumption of seven authentication primitives on five selected architectures. While the SipHash implementation performs best, the HMAC-SHA256 primitive is almost always the slowest. Note the different time scale on the significantly slower embedded real-time processor ARM Cortex-R5 of the brainstem.

distort the overall performance of a given primitive in the road vehicle. For example, we observed Blake2 performing roughly ten times faster if SSE registers were available. Similarly, we doubled the performance of Blake3 using the AVX-512 instruction set. However, especially embedded control units are not necessarily equipped with such hardware extensions. We measure the time it takes for each cryptographic primitive to process test frames of size 100 and 1440 bytes on the selected control units. For more robust results, we repeat each measurement 10,000 times. Figure 6.10 visualizes the time it took to authenticate test packets of size 1440 bytes.

We rank each primitive according to its timing behavior on a particular control unit, whereas a smaller score indicates a better timing behavior. In the end, we sum up the ranks for each primitive to an overall score as shown in Table 6.2. We consider those algorithms the winner with the smallest score in their category. Hence, Blake2 performs the fastest on the sensor modules and the HMI, most likely due to an optimized NEON implementation. In contrast, it ranges among the three worst-performing algorithms on the Cerebrum, although SSE acceleration has been enabled. Therefore, Blake2 occupies only the fourth position among the rated algorithms. Surprisingly, Blake3 beats Blake2 on the Cerebrum but is by far the slowest authentication primitive on the Perception and the HMI. HMAC-SHA256 comes off worst on almost all control units, while the Skein primitive ranges in the midfield. SipHash, Poly1305, and Chaskey perform equally well without significant outliers. Since SipHash has a slightly smaller overall score than the others, we consider it the winner of the authentication category. Similar to the evaluation experiment presented in Figure 6.10, we evaluated the encryption primitives and the elliptic curves. As a result, we deploy *SipHash* for authenticating and *Speck64/128* for encrypting ASOA traffic, while the elliptic curve *Curve25519* is used for efficient key computations as part of the token exchange protocol.

Runtime Security Overhead

Having determined the best-performing cryptographic primitives, we are now ready to assess the total runtime overhead imposed by our security process. To still obtain meaningful and

	Frame Length: 1440 bytes	Brainstem A53	Brainstem R5	Cerebrum	Perception	HMI	Σ Ranks
Encryption	AES-ECB	4	4	3	3	4	18
	Chacha20	2	1	1	2	2	8
	Speck64/128	**1**	**2**	**2**	**1**	**1**	**7**
	Sparx64/128	3	3	4	4	3	17
Authentication	Blake2	6	4	5	1	1	17
	Blake3	5	6	4	7	7	29
	Chaskey	2	1	2	4	4	13
	HMAC-SHA256	7	7	7	6	6	33
	Poly1305	3	2	3	2	2	12
	SipHash	**1**	**3**	**1**	**3**	**3**	**11**
	Skein	4	5	6	5	5	25
EC	FourQ	1	1	2	2	2	8
	Curve25519	**2**	**2**	**1**	**1**	**1**	**7**

Table 6.2: We rank each cryptographic primitive according to its time consumption. The ranks are added to a final score (last column), indicating the overall performance across the selected five control units. The fastest primitives (in bold) are deployed in the road vehicle.

representative results, we measure the time it takes for vehicle services to communicate through the ASOA framework, once with our security extension enabled and once without it. Since the services running on the reference vehicle are protected intellectual property, they are only available in a binary format. Therefore, time measurement is not possible directly at the services' guarantees and requirements because this would require white-box evaluation. While the services internals remain hidden, their interfaces, however, are publicly available in the design tool, which we enhanced in Section 6.2.4. Hence, we can rebuild the skeleton services of the main vehicle event chain. That means our services have the same interfaces but are internally empty, i.e., they do not perform automotive tasks. As an example, the replica of the perception service does not compute an environmental model. Instead, it directly applies the receiving input data to its outputs without further, potentially time-consuming, computations. Hence, the measured relative overhead of our security solution refers to the bare ASOA implementation, whose resource consumption mainly reflects management tasks. Consequently, we expect the relative overhead to drop significantly once services perform computationally demanding tasks. As the relative overhead is likely much lower, we also provide the absolute time for better assessment. To determine the security overhead, we measure the round-trip time to exchange test packets between services running on five selected control units communicating in the reference vehicles. Hence, we claim that the evaluated communication flows are authentic, although the services only perform dummy operations. In total, we conducted three experiments for each combination of control units. That is, we first send test data through raw sockets (UDP), i.e., remove the ASOA framework and, thus, obtain a baseline given by the operating system. Next, we install the ASOA on all ECUs and use it to transfer the same test data. Finally, we activate the proposed runtime protection unit using the fastest-performing algorithms from Table 6.2. As explained earlier, the security level of each dataflow indicates how it is supposed to be secured, i.e., whether the data requires authentic or additional confidential transmission. According to the annotations of the ASOA dataflows, most require authentication but no encryption, probably because most

automotive data is not confidential. Therefore, we use the *AUTH* level for all dataflows in our evaluation setup. In total, we conducted the three mentioned experiments ten times on different control units. As a first finding, the security overhead does not significantly depend on the payload size, although cryptographic operations last longer when the data to be processed grows. Meanwhile, however, the ASOA framework also requires more time to process the data and, in that way, keeps the relative overhead at bay. Second, we observe a more significant overhead (9.56%) once the embedded Cortex-R5 system is involved. One possible explanation is the optimized implementation of the RTPS protocol for embedded systems (c.f., Section 6.1), resulting in more efficient and, thus, faster processing. This assumption is strengthened because the ASOA overhead is comparably low at 23.8%. Note that we expect the relative overhead to drop significantly once services perform actual automotive tasks, but the security overhead remains constant. According to the vehicle design tool's service specifications, the largest service period is 100 Hz. That means the service expects new data to arrive every 10ms. Therefore, we conclude that an average absolute delay of 33.3 µs is acceptable, and our runtime protection is indeed applicable to a vehicular system.

6.4.4 PUF-enhanced Token Exchange Protocol

Ultimately, we evaluate the implementation of our token exchange protocol and primarily focus on control units without the capability to securely store a long-term identity key. In that case, we suggest using a PUF as an identity proof as outlined in Section 6.2.5. Our objective is neither the development nor the improvement of a specific PUF. Instead, we aim to show how unclonable and unique hardware fingerprints apply to the ASOA framework and, thus, to the automotive domain. Our implementation targets the brainstem, the core control unit of the reference vehicle. It is made of a Zynq Ultrascale+ ZU3EG-1E MPSoC that includes an intrinsic FPGA that we leverage to realize a Ring-Oscillator PUF (RO-PUF), a well-established type of PUF [106, 107].

An RO-PUF leverages the frequency differences of multiple ring oscillators to generate a unique fingerprint. We find it particularly well-suited for demonstration because it can be realized on an FPGA. Unlike the brainstem, we acknowledge that most embedded systems do not have an in-built FPGA. In that case, however, other PUF techniques like SRAM [97] or memristor arrays [108] are also reasonable solutions for our token exchange protocol. Figure 6.11 visualizes the basic architecture of an RO-PUF. A single ring oscillator consists of an odd number of k inverters connected in series with a loopback. This structure makes the ring oscillator toggle between one and zero as soon as a voltage is applied. Since each inverter has a unique propagation delay, the toggling frequency of the oscillator is also unique. In total, there are $2n$ ring oscillators, whereas n are connected to a multiplexer. Each multiplexer selects the frequency of a specific oscillator and forwards it to an individual counter. The counter value shows how often the selected oscillator produced a bit. Finally, the counter values are compared, resulting in one bit of the PUF response. Hence, the unique frequency differences of the selected oscillators impact the counter value and, therefore, lead to a unique and unpredictable bit, the PUF response. Depending on the number of required bits, the described construction can be replicated such that longer responses (i.e., bitstreams) become possible. That means an n-bit response requires at least $\lceil \ln(n) \rceil$ different pairs of ring oscillators. In turn, selecting oscillators, i.e., the multiplexers' input, is considered the PUF challenge.

FPGAs are well suited to implement RO-PUFs because of their configurable lookup tables, which are arranged in a matrix structure. Typically, only a small fraction of the FPGA is utilized for

ECU 1	ECU 2	Payload (Bytes)	Raw Sockets		ASOA (vanilla)		ASOA Overhead	ASOA+Security		Security Overhead	
			Mean	Median	Mean	Median		Mean	Median	%	Absolute
Brainstem	HMI	120	215 µs	212 µs	320 µs	318 µs	33.3%	327 µs	328 µs	3.05%	10 µs
		1600	270 µs	278 µs	465 µs	461 µs	39.70%	475 µs	479 µs	2.11%	18 µs
Cerebrum	Brainstem	120	121 µs	119 µs	144 µs	138 µs	12.32%	154 µs	141 µs	2.13%	3µs
		1600	200 µs	200 µs	405 µs	402 µs	50.02%	413 µs	413 µs	2.66%	11 µs
Brainstem	Cerebrum	120	342 µs	351 µs	500 µs	492 µs	28.66%	540 µs	526 µs	6.45%	34 µs
		1600	511 µs	519 µs	677 µs	681 µs	23.8%	761 µs	753 µs	9.56%	72µs
HMI	Door Management	120	374 µs	373 µs	540 µs	535 µs	30.30%	571 µs	570 µs	5.54%	35 µs
		1600	644 µs	642 µs	890 µs	896 µs	28.34%	971 µs	970 µs	7.62%	74 µs
Perception	Cerebrum	120	445 µs	451 µs	765 µs	807 µs	58.17%	811 µs	835 µs	6.01%	28 µs
		1600	568 µs	590 µs	895 µs	912 µs	57.57%	945 µs	960 µs	5.58%	48 µs
					ASOA Overhead:		**36.22%**	**Security Overhead:**		**5.71%**	**33.3 µs**

Table 6.3: We determined the overall runtime overhead of phase P3 of our security process by measuring the time it takes to exchange test packets between five selected architectures. We also compare the original ASOA implementation to raw sockets. Eventually, we average the relative overheads to obtain a single representative number.

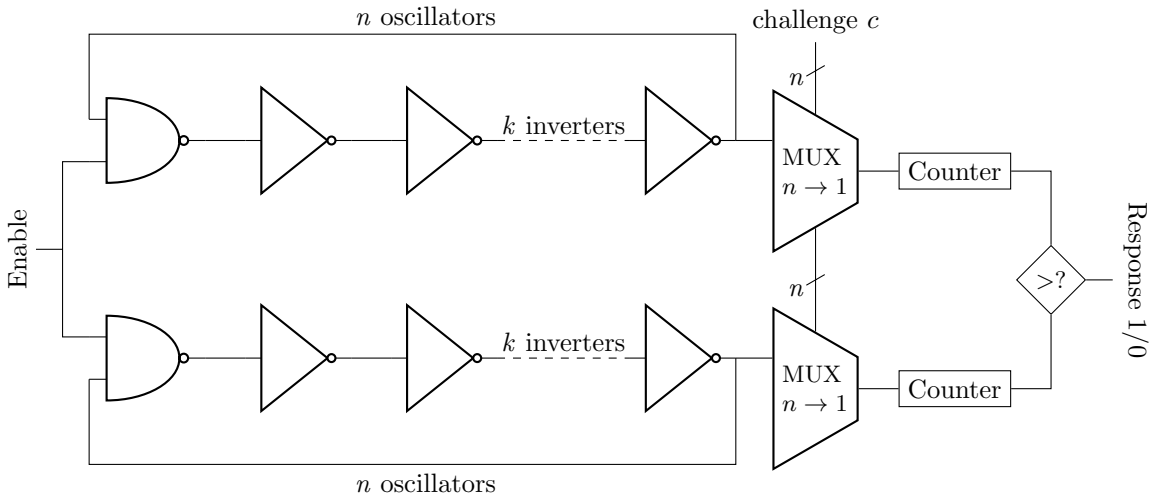

Figure 6.11: An RO-PUF compares the oscillation frequencies of two selected ring oscillators. It generates one bit of the response based on which counter provides the higher value.

specific actions, leaving much space available. This remaining space can be leveraged to manually position ring oscillators with symmetrical line lengths, resulting in a sizable challenge-response space.

To construct our RO-PUF, we employed the Xilinx Vivado Suite[3] to describe the design in Verilog and connect it to the ASOA framework. Figure 6.12 shows our setup as a Vivado block design, where the PUF implementation is represented by the *puf_core* block on the right. The block

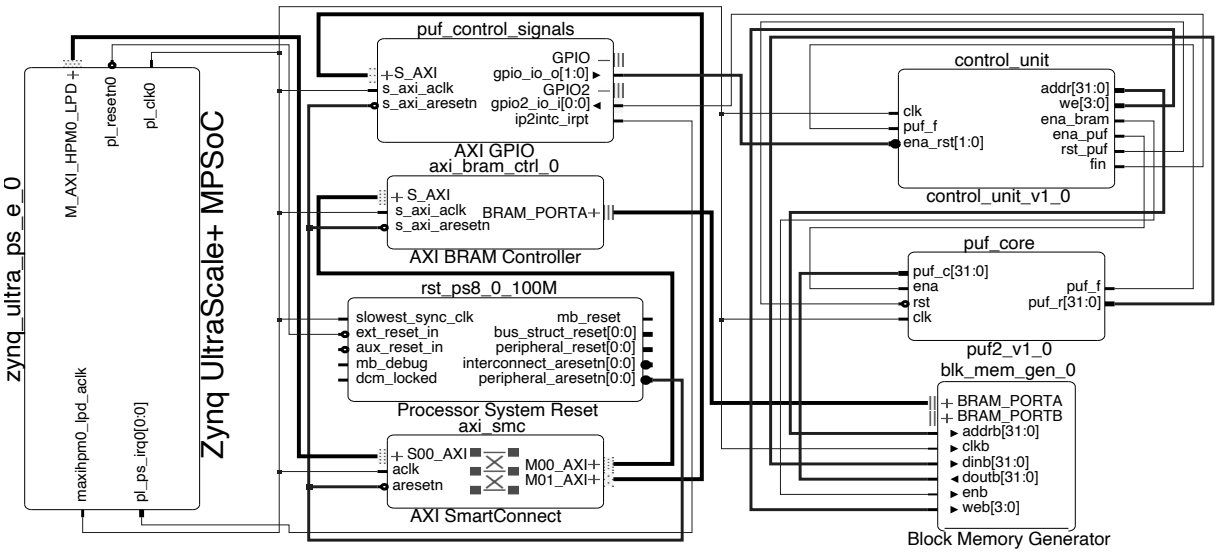

Figure 6.12: The block design from Xilinx Vivado shows how the PUF interacts through a control unit and block memory with the ASOA framework running on the Zynq Ultrascale+ MPSoC.

memory (*blk_mem_gen_0*) beneath the *puf_core* facilitates data sharing between the Cortex-R5 processor and the FPGA. In addition, we introduced a control unit (*control_unit*) positioned above the *puf_core*. It offers three signals through which the ASOA framework can control the PUF.

[3] `https://www.xilinx.com/products/design-tools/vivado.html`

The `Enable` signal is activated once the PUF challenge has been written to the earlier-mentioned block memory. Typically, this happens whenever the ASOA, running in a FreeRTOS environment on the Cortex-R5 processor, requires an identity proof within the token exchange protocol. The `Enable` signal indicates to the PUF that it can proceed to read the challenge, produce a response, and write it back to the block memory. Subsequently, the PUF sets the `Finish` signal, which triggers an interrupt on the application layer. This interrupt notifies the ASOA framework that the identity proof can now be read from memory and processed by the token exchange protocol (c.f., Figure 6.7).

On the opposite side, we implemented the ASP as a threaded server capable of handling multiple incoming requests simultaneously. We successfully executed the token exchange protocol on the Cortex-R5 processor of the brainstem, using the above RO-PUF as identity proof. In total, it took the protocol 15 ms to complete, with approximately 1ms allocated to FPGA request and interrupt processing. Since the security tokens are exchanged during session startup, typically when the vehicle starts, we consider the measured delay acceptable, as no specific time constraints exist in this phase.

6.5 Sub-conclusion

This chapter presented a security process for the Automotive Service-Oriented Architecture (ASOA) to protect the underlying communication from manipulation and data theft attacks. Our solution is transparent to service developers and aligns with ASOA's efforts of providing a centrally manageable software architecture consisting of fully decoupled services. We achieve this by enabling system designers to specify security objectives in a central architecture design tool, which are processed by the ASOA Security Platform (ASP) on session startup. The ASP creates security tokens accordingly and securely transmits them to the control units. That way, changes to the security model are instantly and securely propagated from the design layer to the services inside the vehicle, making the maintenance and update process cost-efficient and simple.

To uphold ASOA's support for embedded and resource-constrained control units, we suggest leveraging distinct physical properties as an identity proof within our token distribution protocol instead of solely relying on secure hardware modules. In this context, we demonstrated the applicability of a Physical Unclonable Function (PUF) as part of our security process in a prototype implementation on an FPGA. Moreover, we organize cryptographic keys in a tree structure to allow a trusted in-vehicle ECU to derive keys for shielded control units computationally, e.g., for those that cannot directly communicate the ASP. We formally verified our token distribution protocol using the Tamarin model checker and thoroughly evaluated our work in a self-driving prototype vehicle. Our experiments showed that our work imposes an average runtime overhead of 5.71% corresponding to an absolute delay of 33.3 µs.

To answer research question RQ3, we conclude that an efficient security process should allow the seamless integration of security aspects without introducing dependencies between software modules. It should support changes to the software architecture, requiring updating permissions, cryptographic keys, and security levels. Lastly, it should enable the collection of logs and analysis of security incidents. Our work contributes to a holistic security approach since the proposed security process begins in the vehicle design phase and allows for modifications when the vehicle is actively used.

7. Ensuring Software Integrity of ECUs

Section 3.1 has illustrated how well-known cyberattacks on vehicles operate and has made it clear that not only unprotected communication but also corrupted ECUs poses a serious threat to automotive systems. This fact aligns with our security requirement analysis from Chapter 4, which states that while secure communication is an essential part of a security concept for an SDV, yet control units and their software must also be protected from manipulation. This is especially necessary when they can be updated remotely or even allow user customization, such as personalizing an infotainment system.

Consequently, the requirement "system integrity" includes not only communication but also software integrity. Otherwise, attackers not sitting between ECUs but on them could take control of the vehicle and provoke fatal consequences.

This chapter addresses how the software state of SDVs can be quantified, verified, and validated. The objective is to prove the trustworthiness and thus the roadworthiness of an SDV to a third party. If verification and validation are successful, the examined vehicle receives a digital certificate authorizing it to participate in traffic or other services. Since compromised control software can have fatal safety consequences, similar to faulty or worn mechanical components, we argue integrity checks are essential for SDVs. Our work may contribute to regulations requiring proof of uncompromised and trustworthy software, similar to mandatory safety and quality inspections of mechanical vehicle parts. We compare this with the legally required technical inspection of mechanical components.

Unlike the previous chapters, we assume an attacker residing on the ECUs. To detect this, we present a proactive scheme for measuring the software integrity of ECUs. In this context, we first describe the system model in Chapter 7.1, then specify the attacker model, and finally present a software validation scheme to answer the research question RQ4 in Section 7.3. Section 7.5 describes our implementation, which we then evaluate in Section 7.6.

This chapter builds upon the research described in the article "Ensuring Trustworthy Automated Road Vehicles: A Software Integrity Validation Approach" [6], which was presented at the IEEE International Automated Vehicle Validation Conference in 2023. I thank Felix Klement for supporting the evaluation of the registration overhead in Section 7.6.2.

7.1 System Model

Each vehicle within our ecosystem is assumed to possess a unique digital identifier called *VID*, akin to a license plate number. Furthermore, we assume the existence of a central ECU, denoted as $\mathcal{E}_C$, which, aside from providing computational power for automotive tasks, functions as a communication gateway for external components.

We introduce a generic Registration Unit (RU) requiring vehicles to register for actions they intend to perform. The idea behind the RU is to grant permission to the requested action only if the vehicle can provide proof of trustworthiness, thereby ensuring a validated software state, as compromise may pose a significant hazard to other traffic members. The precise role of the RU can be customized to suit a variety of mobility concepts. For instance, a vehicle may wish to participate in automated traffic in an urban environment, where the RU could comprise all automobiles in a specified perimeter that share environmental sensor data and, thus, need to trust each other. Alternatively, the RU could be a traffic surveillance unit, as has been suggested by recent research [109] and even required by a German draft law [92] from 2019. Another potential use case may be a platoon whose master vehicle requires proof of trustworthiness from vehicles seeking to join.

In our system, a vehicle's trustworthiness is attested by the Trusted Software Authority (TSU), another vehicle-external component. It is primarily responsible for the maintenance of automotive software and, for that purpose, keeps a copy of each authentic software component. The TSU acts as a *verifier* and issues a certificate if the vehicle can prove its benign software state. Notably, unlike related work [8] that assumes an in-vehicle trust anchor, our approach entails each vehicle solely trusting the remote TSU and possessing its public key. Furthermore, we assume the existence of a shared identity key, denoted as $K_{\mathcal{E}_i}$, between each ECU and the TSU.

We acknowledge that maintaining a separate key for each ECU in every vehicle does not scale in a large automotive ecosystem. Therefore, envision a key derivation mechanism, allowing the TSU to computationally determine $K_{\mathcal{E}_i}$ based on the vehicle identity *VID*. However, a practical solution is orthogonal to the problem of ensuring trustworthiness and is left for future research.

7.2 Attacker Model

We assume an adversary, denoted as $\mathcal{A}dv$, who possesses control of the in-vehicle and external network (Dolev-Yao) and the ECUs. $\mathcal{A}dv$ can drop, inject, manipulate, and delay messages transmitted within the vehicle as well as traffic to external units. Additionally, $\mathcal{A}dv$ can penetrate ECUs, where he may install potentially harmful software and execute it, facilitated through over-the-air updates or poorly secured interfaces. However, we do not consider the manipulation of software during vehicle operating time. While $\mathcal{A}dv$ can infiltrate hardware, he cannot affect the code execution within trusted environments such as Intel Software Guard Extensions (SGX) or ARM TrustZone. Moreover, $\mathcal{A}dv$ cannot break cryptographic primitives as he is computationally constrained.

7.3 Software Validation Scheme

Our scheme for validating vehicle integrity comprises five consecutive steps, executed every time proof of trustworthiness is required, typically when a vehicle starts. The fundamental principle involves the computation of integrity identifiers for every ECU in a cascading manner, considering the integrity of that ECU and that of its children. Eventually, a trustworthy vehicle receives a certificate that enables it to register for an automotive action, such as participating in urban traffic or joining a platoon. Figure 7.1 illustrates our scheme schematically.

7.3.1 Initialization

The integrity measurement begins with the central ECU $\mathcal{E}_{\mathrm{C}}$ sending an attestation request to the TSU, containing the vehicle's identity *VID*. In response, the TSU generates a random nonce n and a distinct attestation key $k_{\mathcal{AT}}$, which will be used to authenticate the integrity measurements. The purpose of n is to ensure freshness and thwart $\mathcal{A}dv$ from replaying outdated measurements. To prevent leakage of $k_{\mathcal{AT}}$, the TSU encrypts it for each ECU using the corresponding identity key $K_{\mathcal{E}_{\mathrm{i}}}$ to obtain $\mathsf{C}_{\mathcal{E}_{\mathrm{i}}}^{k_{\mathcal{AT}}} = Enc(k_{\mathcal{AT}})_{K_{\mathcal{E}_{\mathrm{i}}}}$. The encrypted keys $\mathsf{C}_{\mathcal{E}_1}^{k_{\mathcal{AT}}}, \mathsf{C}_{\mathcal{E}_2}^{k_{\mathcal{AT}}}, ..., \mathsf{C}_{\mathcal{E}_{\mathrm{n}}}^{k_{\mathcal{AT}}}$ along with n are sent back to the vehicle, where $\mathcal{E}_{\mathrm{C}}$ disseminates them to all $\mathcal{E}_{\mathrm{i}}$.

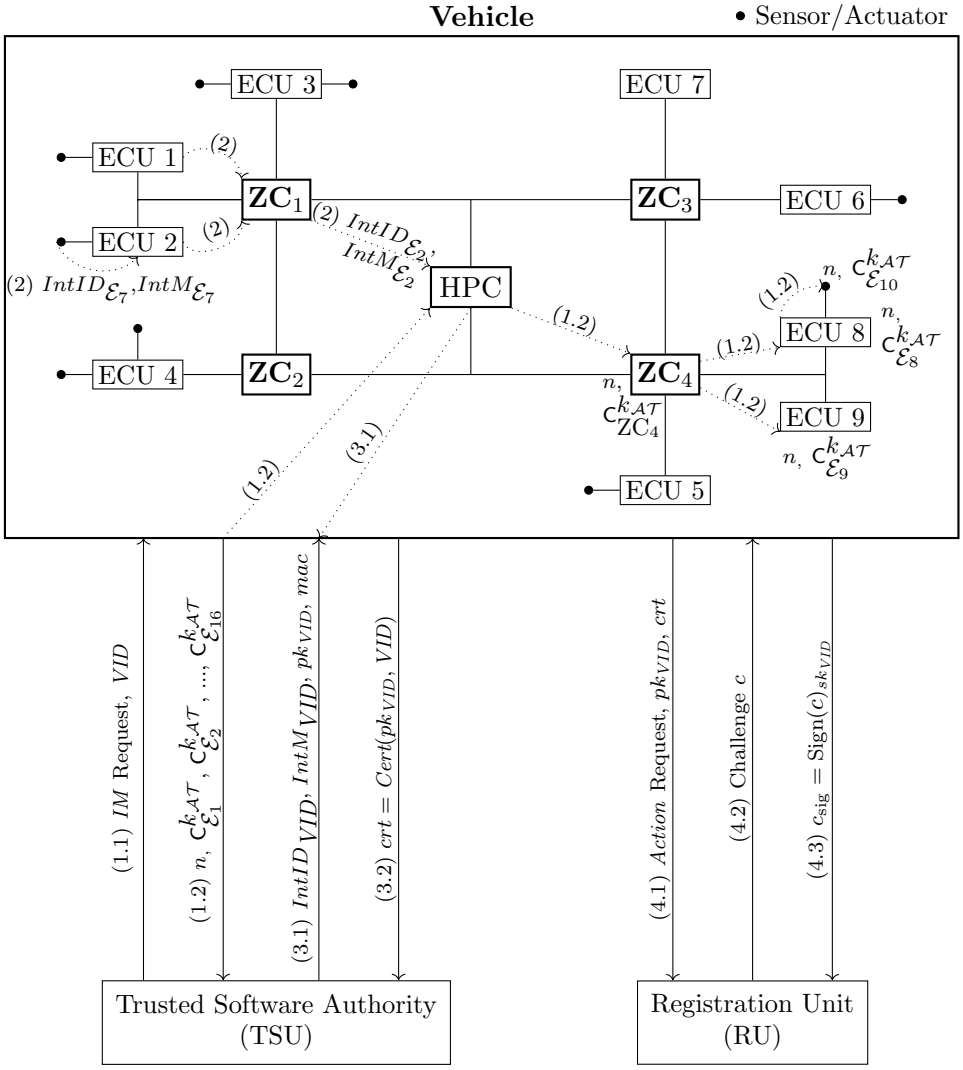

Figure 7.1: An illustration of our integrity validation scheme in a simplified zonal architecture with a central HPC and four zones. After distributing a nonce and an encrypted attestation key to each ECU (Step 1.2), the ECUs determine their software integrity by running *Measure* (Step 2) in a secure hardware-isolated environment. The integrity identifier is sent, along with the measurements, to the TSU for validation (Step 3.1). Depending on the result, the TSU issues a certificate (Step 3.2), which proves trustworthiness to an RU, allowing to establish a secure channel required to perform the requested action (Step 4). To enhance legibility, we only show selected steps in one zone each.

7.3.2 Integrity Measurement

Upon receipt, $\mathcal{E}_{\mathrm{i}}$ commences the process of measuring its software integrity by calling the procedure *Measure* (Algorithm 2) in a secure environment. The measurement of $\mathcal{E}_{\mathrm{i}}$ produces an

integrity identifier that we refer to as $IntID_{\mathcal{E}_i}$. This value is the authenticated hash of selected software components, denoted as $\mathsf{SC}_{\mathcal{E}_i}$. These components encompass software on the functional layer, such as SOME/IP or ASOA services, ROS nodes, or custom binaries, often provided by third parties, as they are subject to regular extension and updates. To prevent $\mathcal{A}dv$ from penetrating lower levels of the software stack, including the OS or the bootloader, we also ensure the integrity of system files and the bootloader during *authenticated boot* (c.f., Section 7.4.1).

Measure operates in a cascading manner, merging the measurement of the current ECU with that of its children. As a result, $IntID_{\mathcal{E}_i}$ represents the integrity of $\mathcal{E}_i$ and all $\mathcal{E}_j$ that are positioned hierarchically below. This enables the inclusion of shielded ECUs, such safety-critical embedded systems that are directly wired to their parent without the ability to communicate with external components. For example, if $\mathcal{E}_i$ was a zone controller, $IntID_{\mathcal{E}_i}$ would incorporate not only the integrity of $\mathcal{E}_i$ but also of all ECUs within that particular zone.

Algorithm 2 Software Integrity Measurement

```
1:  procedure MEASURE(SC_Ei, IntID^C_Ei, IntM^C_Ei, C^{k_AT}_Ei, n)
2:      IntM ← array()                                  ▷ contains all measurements
3:      k_AT ← Dec(C^{k_AT}_Ei)_{K_Ei}                  ▷ get attestation key
4:      for each sw ∈ SC_Ei do                          ▷ measure sw integrity
5:          bin = Load(sw)
6:          hash ← Hash(bin)                            ▷ e.g., using Linux IMA
7:          IntM[E_i].Append(sw, hash)
8:      end for
9:      authboot ← GetAuthBootCodes()
10:     IntM[E_i].Append("bootchain", authboot)
11:     while idx ≠ len(IntID^C_Ei)) do                 ▷ verify authenticity
12:         IntID_Ej ← IntID^C_Ei[idx]
13:         IntM_Ej ← IntM^C_Ei[idx]
14:         if Verify(IntID_Ej)_{k_AT} = Authentic then
15:             IntM.Join(IntM_Ej)
16:         else
17:             IntM[E_j] ← "untrusted"
18:         end if
19:     end while
20:     IntID ← MAC(IntM, n)_{k_AT}                     ▷ authenticate
21:     SendToParent(IntID, IntM)
22: end procedure
```

Along with $IntID_{\mathcal{E}_i}$, *Measure* produces an array $IntM$ that contains the labels of the measured software and the corresponding hashes. $IntM$ enables the TSU to identify software whose integrity check failed and factor this during validation. Given that an ECU may have several children, we utilize the array $IntM^C_{\mathcal{E}_i}$ to store the $IntM$ of $\mathcal{E}_i$'s children. Similarly, $IntID^C_{\mathcal{E}_i}$ holds the integrity identifiers of $\mathcal{E}_i$'s children. In line 3, *Measure* first decrypts $\mathsf{C}^{k_{\mathcal{AT}}}_{\mathcal{E}_i}$ to receive the attestation key $k_{\mathcal{AT}}$. Then, starting in line 4, *Measure* performs integrity measurement by computing the hash of each software, which is added along with the software label to $IntM_{\mathcal{E}_i}$ in line 7. Following this, *Measure* retrieves the results from the authenticated boot (line 9) to $IntM[\mathcal{E}_i]$, allowing the TSU to verify the benign state of the OS and the bootloader. Finally, *Measure* verifies the authenticity of the measurements (line 11) carried out by $\mathcal{E}_i$'s children using the key $k_{\mathcal{AT}}$. This step is necessary to exclude manipulations of the measurements during their

transmission within the vehicle. If the verification succeeds, the hashes and software labels of $\mathcal{E}_\text{i}$'s children are appended to $IntM_{\mathcal{E}_\text{i}}$. Otherwise, $\mathcal{E}_\text{i}$ flags the software of the respective ECU as "untrusted", allowing the TSU to determine potential consequences during validation.

Ultimately, in line 20, the integrity identifier $IntID$ for ECU $\mathcal{E}_\text{i}$ is computed using a MAC that takes as input the hashes and software labels in $IntM$, along with the nonce n. Since $IntM$ comprises the measurement of $\mathcal{E}_\text{i}$'s children, the $IntID$ also reflects their integrity state. Finally, both the $IntID$ and the corresponding measurements stored in $IntM$ are sent to the parent ECU (line 21).

7.3.3 Validation

A trustworthy vehicle receives a certified digital identity that enables it to register with the RU. This certificate links the vehicle's identity VID to a public key pk_{VID}, which is generated by $\mathcal{E}_\text{C}$ alongside the corresponding secret key sk_{VID} every time the integrity measurement process terminates. Since $\mathcal{E}_\text{C}$ has no parent ECU within the vehicle, it sends $IntID_{VID}$, $IntM_{VID}$, and pk_{VID} to the TSU after terminating *Measure*. Furthermore, it transmits $mac = MAC(VID, pk_{VID})_{k_{\mathcal{AT}}}$, allowing the TSU to verify the authentic origin of pk_{VID}. Upon reception, the TSU first verifies mac and $IntID_{VID}$. For that purpose, the TSU extracts the software labels from $IntM$, queries the original binaries from its database, and computes their hashes. The vehicle's software state is considered fully trusted if the verification succeeds immediately. Otherwise, the TSU identifies the distrusted software and validates whether it must be considered critical. We propose implementing a validation process that considers the ECU on which the untrusted software runs and whether it has access to actuators and the potential to impact vehicle dynamics. Customization at or below the OS must be strictly prohibited. Therefore, the integrity of all boot stages must be ensured. That way, a customized application operating on the infotainment system may not pose a threat, while unverifiable software in a safety-critical zone must inevitably lead to countermeasures. If the validation judges the vehicle trustworthy, the TSU issues a certificate $crt = Cert(pk_{VID}, VID)$, which is eventually sent back to the vehicle. Otherwise, an error is returned, and consequently, pk_{VID} remains uncertified, rendering it impossible for the vehicle to register with the RU.

7.3.4 Registration

Registering a vehicle with the RU entails applying for a specific automotive action, which may vary depending on the mobility concept being employed. In either case, a secure communication channel is necessary to perform the requested action, which gives the possibility to exclude illegitimate vehicles. The registration is a challenge-response mechanism between a vehicle and the RU. Apart from the desired action, the request contains the vehicle's public key pk_{VID} and the previously obtained certificate crt. Once the request is received, the RU verifies the certificate and responds with a challenge, denoted as c. Upon reception, the requesting vehicle computes the signature $c_\text{sig} = \text{Sign}(c)_{sk_{VID}}$ and sends it back to the RU. If the RU successfully verifies c_sig, the vehicle is considered trustworthy, and the requested action is permitted by establishing a secure channel between the RU. This is achieved through the use of the trusted public key pk_{VID}, which may be employed, for example, through a Diffie-Hellman key exchange. Otherwise, the request is rejected, and the vehicle is excluded from further actions. Hence, successful registration is contingent upon the vehicle having undergone an integrity measurement process prior to the request. Otherwise, the authenticity of pk_{VID} cannot be proven to the RU as the certificate crt would not have been issued.

7.3.5 Invalidation

To prevent $\mathcal{A}dv$ from using certificates representing an outdated software state, the TSU revokes the issued certificate when a vehicle is powered off. To achieve this, the central ECU $\mathcal{E}_{\mathrm{C}}$ implements a notification procedure *Notify* to inform the TSU when the vehicle is shut down. Similar to *Measure*, this procedure is executed in a secure environment.

7.4 Security Requirements

The genuine and secure execution of *Measure* and *Notify* is the fundamental assumption of our scheme. The problem, however, is that these functions are executed in a possibly compromised environment since our attacker model assumes the presence of $\mathcal{A}dv$ on the ECUs. Therefore, both procedures and the secret identity key $K_{\mathcal{E}_{\mathrm{i}}}$ are part of the Trusted Computing Base (TCB), which is inaccessible to potential adversaries. This design ensures that even if $\mathcal{A}dv$ gains access to an ECU, it is impossible to tamper with or prevent *Measure* from executing as intended, i.e., it must not be interrupted by any other process. Technically, the TCB is protected through enclaves, which provide an isolated and secure environment for running trusted code, commonly referred to as *secure world*. As elaborated in [8], *Measure* and *Notify* must be kept in a write-protected memory area to preclude tampering. Similarly, the secret identity key $K_{\mathcal{E}_{\mathrm{i}}}$ must be safeguarded against illegitimate access, which can be achieved through secure memory. Exclusive access to $K_{\mathcal{E}_{\mathrm{i}}}$ can be enforced using an I/O Memory Management Unit.

7.4.1 Authenticated Boot

Beyond applications at the functional layer, it is essential to incorporate low-level software components into the integrity measurement process, particularly if ECUs are equipped with a full-stack OS. $\mathcal{A}dv$ may compromise components such as the bootloader to gain control of the ECU. To address this, we employ authenticated boot, a technique to ensure that a computer system loads in an expected and trustworthy state. The boot process typically involves multiple stages that form a chain. The process usually begins with a code snippet from read-only memory that contains the logic to select the boot device where the First-Stage Bootloader (FSBL) is expected. The FSBL initializes basic hardware controllers and loads the Second-Stage Bootloader, for example, a U-Boot environment. The latter loads the Linux kernel space, which in turn loads the Linux user space where automotive applications run. During authenticated boot, the integrity of each stage of this boot chain is measured. The measurement results are stored in memory rather than verified immediately, as is the case with *secure boot*. We retrieve these results in line 9 of Algorithm 2 and integrate them into the integrity identifier as a single hash.

7.4.2 Remote Attestation

For measuring the software integrity on a possibly compromised ECU, we leverage the principles of remote attestation that have been widely discussed [110]. Typically, a *prover* makes a claim about its system to a remote party, also referred to as the *verifier*. During attestation, the prover provides computational evidence about this claim to the verifier, who either confirms or rejects it. Special hardware extensions like a Trusted Platform Module, Intel SGX, or the ARM TrustZone provide an isolated execution environment in which the prover creates the evidence. Alternatively, lightweight solutions [111, 112] can be used for embedded systems. In our work, we leverage the technique of remote attestation to execute *Measure* and *Notify*.

7.5 Implementation

We implemented and evaluated the proposed integrity measurement scheme on the hardware used in our reference vehicle presented in Section 4.1. As illustrated in Table 4.1, many ECUs employ ARM processors, which benefit from their inherent safety and security features. For instance, modern ARM processors support a dual-core lockstep mode and use the TrustZone technology for software isolation. In our setup, we recreated the zonal E/E architecture of the reference vehicles in a best-effort approach using the original hardware where possible. That means we utilized the brainstem's Ultrascale platform for the central ECU $\mathcal{E}_C$ and RPis for the remaining controllers. To implement the proposed scheme, each ECU needs to know its parent and child nodes within the E/E architecture. Figure 7.2 schematically illustrates the hierarchical arrangement of the ECUs. Each zone ultimately connects to the central brainstem and consists of

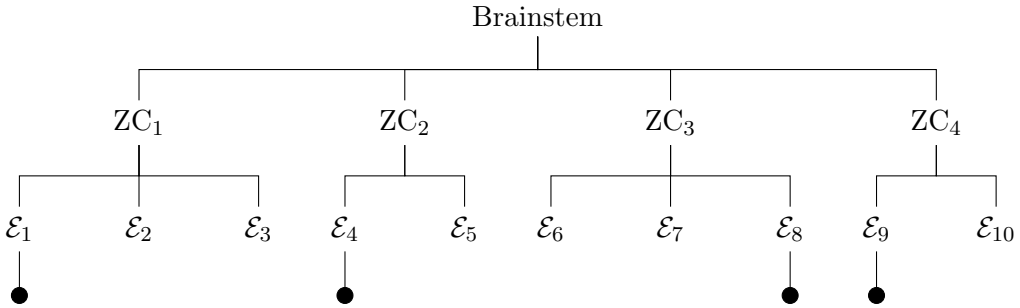

Figure 7.2: The zonal E/E architecture replicated by our setup consists of four hierarchical layers: (1) the central brainstem as the root node, (2) the zone controllers, (3) the ECUs, and (4) sensors and actuators.

two or three ECUs, whereas one is connected to a sensor or actuator. In total, our setup consists of 19 ECUs. We use ordinary laptops with Intel Core i7-1260P processors and a full-stack Linux OS for the TSU.

7.5.1 OP-TEE

To ensure compliance with the security requirements outlined in Section 7.4 for the execution of *Measure* and *Notify*, we utilize the publicly available Open Portable Trusted Execution Environment (OP-TEE) [1], a framework leveraging the ARM TrustZone technology to create a *secure world* in which applications and data are isolated and protected from access by the *normal* world. This approach builds upon prior work [8] in which we proposed an attestation scheme for ECUs running a full-stack OS. OP-TEE uses the ARM Security Extensions to establish a Trusted Execution Environment (TEE) inside the secure world using two core building blocks: A Trusted Application (TA) is a signed binary that runs in the secure world and has access to shielded cryptographic features such as secure memory. In contrast, the Secure Monitor (SM) starts and stops TAs and manages communication channels between the secure and normal worlds. Note that while resources in the secure world may access the normal world, the reverse is impossible. In our implementation, we developed the *Measure* and *Notify* procedures in C++ as TAs. *Measure* uses the Linux Integrity Management Architecture (IMA) to perform the hashing of the targeted applications in SC. At the moment, we include *all* files in user space into SC to facilitate the creation of a policy for the IMA kernel module. The specification of SC may be adjusted to include or exclude specific files from processing. To allow ECUs to compute an aggravated integrity identifier, we include the addresses of the child and parent ECUs into each instance of

[1] `https://www.op-tee.org/`

Measure. We argue that hardcoding these addresses seems appropriate as the network topology remains static. The hashes produced by the IMA are securely stored and made available to *Measure* in the TEE for processing them as described in Algorithm 2.

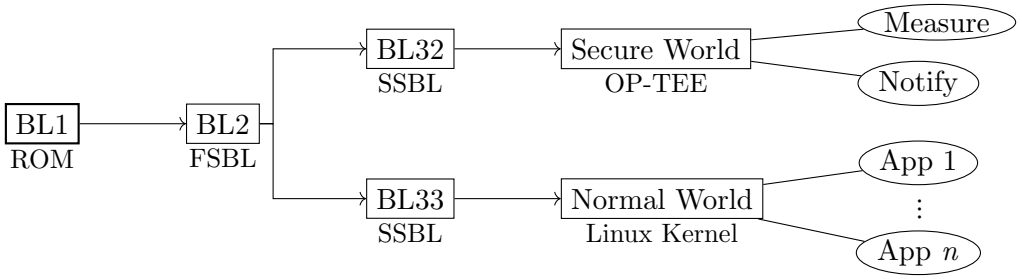

Figure 7.3: The ARM boot chain uses a separate Second-Stage Bootloader to load a secure world, in which we execute the *Measure* function.

To guarantee the integrity of the OP-TEE environment, we integrate it into the authenticated boot process, as illustrated in Figure 7.3. For that purpose, we leverage the ARM boot chain, which provides dedicated stages for loading a TEE within the secure world. Specifically, the BL32 boot stage loads the SM and then the OP-TEE while the BL33 stage loads the Linux kernel in the normal world.

While this setup allows us to evaluate the proposed scheme in a realistic environment, it is important to note that our implementation remains a prototype requiring further engineering efforts for secure deployment in a road vehicle. Notably, the RPi lacks a Memory Protection Unit and hence the ability to provide secure storage, enabling an attacker to access sensitive data such as key material from the normal world. Additionally, our approach currently does not protect against runtime attacks, as automotive applications may be altered after the measurements have been made.

7.6 Evaluation

We utilize our implementation to evaluate the proposed scheme in two steps. Initially, we analyze the average time required for determining and validating the integrity of a vehicle. Subsequently, we investigate the registration overhead by simulating a moving vehicle in both an urban and a highway environment, where it is mandated to prove its benign software state to nearby entities.

7.6.1 Validation Overhead

We began by determining the boot time of each ECU without our scheme enabled to obtain a baseline. This involved loading a bare Linux OS without performing authenticated boot or computing integrity identifiers. We observed an average boot time of 17.2 s on the RPis platforms and 15.3 s on the central Xilinx Ultrascale+ system. Next, we built the OP-TEE environment, deployed it on our setup's ECUs, and initiated the *Measure* procedure in the secure world. During the integrity measurement process, we generated hashes of *all* files stored on the ECUs. Specifically, the RPi platform contains 130,578 system files, while the Xilinx Ultrascale+ board has 48,738 files. Note that the integrity of system files is expressed as a single hash, as described in line 9 of Algorithm 2. Hence, the TSU can recognize an untrusted OS but cannot identify the corrupted files, which is unnecessary as we always require a benign OS. In contrast, automotive

		Bare Boot	Auth. Boot + *Measure*	Overhead
Single ECUs	RPi	17.2 s	21.3 s	4.1 s
	Brainstem	15.3 s	18.4 s	3.1 s
Full Scheme	Tree		600ms	
	Bus		1.8s	
Validation Delay			1036 ms	
Total Averaged Overhead			**5.236 s**	

Table 7.1: Time consumption to validate the software integrity of our setup with each ECU running four automotive applications.

applications in $\mathsf{SC}_{\mathcal{E}_i}$ are individually included in the integrity identifier, as these applications are typically added or updated and, thus, require precise validation. We created four dummy ASOA services for each ECU to simulate these applications. This number originates from the earlier mentioned prototype vehicles, which deploy 4.3 automotive applications on average on every ECU. As shown in Table 7.1, our scheme executes in 21.3 s on the RPi platform and 18.4 s on the Ultrascale+ board. Thus, the integrity measurement adds an overhead of 4.1 s and 3.1 s, respectively, to the original boot process.

While the execution time on a single ECU is a first reasonable performance estimate, we still need to consider the time a given ECU needs to wait for its children to terminate. Recall that an integrity identifier includes the measurements of all hierarchically lower positioned ECUs, resulting in a dependency among them. For instance, the zone controller ZC_1 shown in Figure 7.2 can only proceed computing $IntID_{\mathrm{ZC}_1}$ and $IntM_{\mathrm{ZC}_1}$, after the ECUs $\mathcal{E}_1$, $\mathcal{E}_2$, and $\mathcal{E}_3$ have provided their results.

Our setup takes 21.9 s from booting the system to the computation of the integrity identifier for the ECU on the highest hierarchical layer, i.e., the brainstem. Hence, we observe an additional latency of 600 ms compared to the longest execution of *Measure* on a single device. This additional latency encompasses the network delay and the time of the ECUs for their children to terminate and, therefore, highly depend on the hierarchical structure of the E/E architecture. Ideally, the ECUs are arranged in a flat tree since the integrity measurement can occur fully parallel as only the root node needs to wait. In the worst case, the ECUs are arranged in a chain where each ECU has to wait for the adjoining one. We modified our setup to implement the latter case and observed that the computation of the vehicle's integrity identifier consumes 23.1 s, corresponding to an overhead of 1.8 s. We conclude that the hierarchical arrangement of the ECUs does impact the scheme execution time. However, this overhead is relatively low compared to the time necessary to boot and measure the individual files.

Finally, the vehicle's integrity identifier has to be transmitted to and validated by the TSU. In a real scenario, a cellular V2X channel would probably be used for transmission. For simplicity, we use a wireless connection based on the IEEE 802.11ac standard. We measure the time it takes to transmit the integrity identifier from the central brainstem to the TSU, to validate it, and to reply with a certificate of trustworthiness. In our case, the validation describes the cryptographic verification process, but we do not judge the deployed dummy applications since they only serve as an example. We observe an average time of 1036 ms to perform the earlier-mentioned steps on an authenticated integrity identifier of size 1130 KB.

Parameter	Value
Road layout	[Highway, Urban] 5km, 3+3 lanes
Density	[50, 100, 150, 200] vehicles/km
Ranges of Awareness	[50, 100, 200, 300] meters
Average speeds	[30, 50, 120] km/h (± 12 km/h)
Channel	5.9 GHz
Bandwidth	10 MHz
Antenna gain (tx and rx)	3 dBi
Propagation model	WINNER+, Scenario B1
Shadowing	Variance 3 dB, decorr. dist. 25 m
Registration Payload	2136 KB

Table 7.2: Technical parameters for simulating the vehicle registration phase.

Overall, we have a total average overhead of 5.236 s. At first glance, this figure might seem high; however, most time is necessary for measuring the system files of the unoptimized OS. The overhead can be reduced by appropriately configuring the OS and minimizing its overall size. Considering that this overhead is added to the vehicle booting process, we find it acceptable.

7.6.2 Registration Overhead

At last, we aimed to determine whether the latency of the registration step is acceptable in a real scenario. To answer this question, we simulated an ecosystem in which *traffic entities* engage in continuous communication within a Range of Awareness (RoA) to optimize traffic and improve safety. For example, they might share sensor data to enhance situational awareness, especially in cases where obstacles or events are hidden from their direct perception. A traffic entity describes a communicating automotive object such as a vehicle, a roadside unit, or a traffic symbol. A vehicle seeking to join automated traffic must provide evidence of its trustworthiness to all entities in the RoA, i.e., registration is necessary. In other words, the RU consists of all traffic entities within a given RoA. We used a Matlab simulation framework [113] to examine the average time required for a vehicle to register with traffic entities in its RoA. This framework enabled us to simulate the data transmission based on Dedicated Short Range Communication, which is based on the IEEE 802.11p standard.

We assumed a moving vehicle; thus, new traffic entities appear in the RoA while others leave. Consequently, the registration step was continuously repeated while the vehicle was in motion. Table 7.2 summarizes the configuration parameters of our simulation that considers an urban and a highway scenario, both established by the ETSI [114]. The scenarios mainly differ in the density of obscuring objects, which affects the scattering behavior. The urban environment is densely populated with many roadside units, traffic signals, and vehicles. In contrast, our highway features three lanes with a primarily unobstructed line of sight. Vehicles existing on one end of the highway re-enter the same lane on the opposite one, requiring a new registration process as they are treated untrusted again. We modeled vehicle speeds in the urban and high environments using a Gaussian distribution with average speeds of 30 km/h and 50 km/h, and 90 km/h and 120 km/h, respectively. The standard deviation used for both environments was 12 km/h.

Our simulation encompassed multiple experiments investigating the registration overhead, i.e., the delay from a moving automobile to all traffic entities within its RoA. We considered densities of 50, 100, 150, and 200 traffic entities within a perimeter of one kilometer and RoAs of 50, 100, 200, and 300 meters. Note that the RoA generally has fewer traffic entities as it only covers part of the scenario area. A single registration required the transmission of the certificate of trustworthiness *crt* (1048 KB), the public key pk_{VID} (1024 KB), and the action string (64 B) as described in the fourth step of our scheme. We also add a flat rate of 10 ms to our results to account for the time required to verify *crt* and the challenge signed *c*, acknowledging that this value may vary on different systems.

Figure 7.4 visualizes the relationship between the average traffic density in all RoAs and the average registration overhead. Note that the number of traffic entities within the RoAs is assumed to be the same in the highway and urban scenario. Our results show that the delay grows

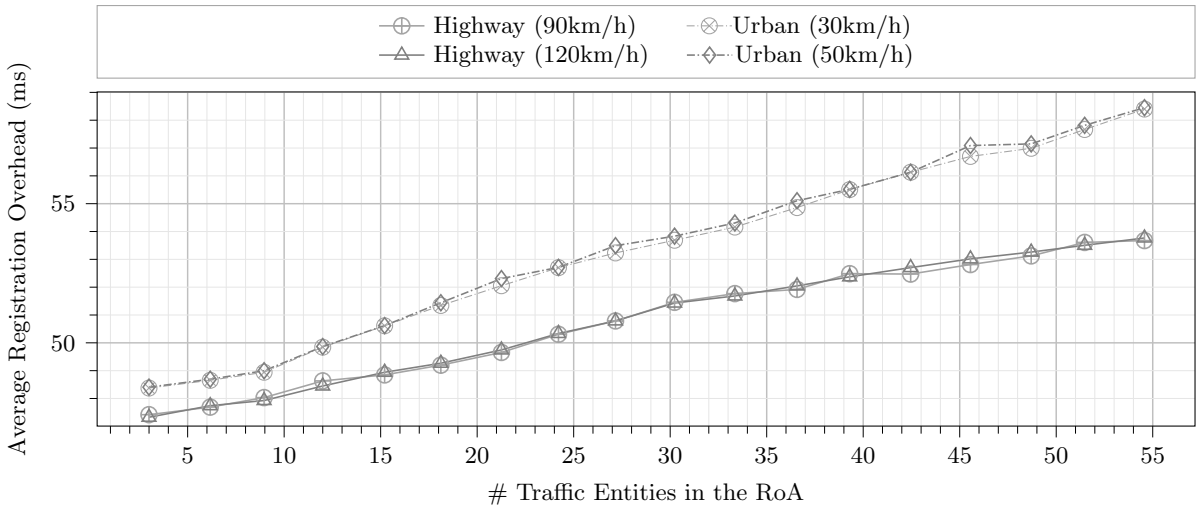

Figure 7.4: The more traffic entities are in the RoA, the longer the registration takes. In an urban ETSI scenario, the registration generally lasts longer due to more obscuring objects than on a highway.

with increasing traffic entities, although registrations are independent of each other and run in parallel; as expected, more communication leads to more channel congestion and a larger delay. Additionally, on average, registration takes longer in an urban environment than on a highway, likely due to more obscuring objects, leading to a different scattering behavior. The registration delay ranges from 47 to 58.5 ms, which we found acceptable as the *minimal*-induced delay in such settings has been estimated to be approximately 40-50 ms [115].

7.7 Sub-conclusion

To give an answer to research question RQ4, we presented a novel scheme for determining and validating the software integrity of road vehicles. The trustworthy software state of SDVs can be proven with regular integrity measurements carried out in isolated environments on ECUs. Unlike existing solutions, we do not rely on a trusted in-vehicle verification unit since such an approach does not scale for modern updatable systems. Instead, our scheme involves a remote component to validate a vehicle's integrity, allowing it to approve uncritical customizations or software from unknown sources. This component issues a certificate if it finds the vehicle's software trustworthy, enabling it to register with third parties requiring evidence of its trustworthiness. That way, other

vehicles, roadside units, or authorities can ensure that vehicles do not pose a safety threat due to malicious software modifications.

We employ hardware isolation techniques to securely compute an integrity identifier on potentially compromised systems in a cascading manner, leveraging the hierarchical structure of modern E/E architectures. We implemented the scheme on a prototype setup based on a recently presented zonal E/E architecture, evaluated its performance, and simulated the registration step in an urban and highway ETSI scenario.

Our experimental results show that the proposed scheme currently incurs an average overhead of 5.236 s to the vehicle's starting procedure, while the average registration delay ranges between 47-58.5 ms.

8. Reaction to Security Incidents

This chapter assumes an SDV that is optimally protected against all known threats, regularly monitors its security status, detects possible attacks, and operates on the road. Therefore, we now look at the operation phase of the vehicle life cycle. Even a well-protected SDV is not completely immune to cyberattacks due to the constantly changing and growing attack surface. With a high probability, security incidents will occur, either intentionally or accidentally. In both cases, we want the SDV to remain operable as long as possible and only initiate an emergency stop if the passengers' safety cannot be ensured anymore. Consequently, SDVs require means to assess and react to security incidents since not each can be attributed to an attack.

For instance, how should the vehicle behave if it detects unauthenticated traffic while in motion? The answer to this question depends on various factors, such as the vehicle's topology, its context (e.g., its current speed), and the deployed security concept. A missing answer, however, can lead to unexpected behavior, such as a compromised infotainment system allowing the acceleration of the vehicle [18]. That means SDVs must be resilient to security incidents.

Our security requirement analysis from Chapter 4 underscores the necessity of monitoring and reacting to security incidents. While this requirement originally does not stand for immediate response to security incidents but rather expects logging and reporting mechanisms, we expand the notion that an SDV can interpret the alarm triggered by any deployed security means.

In this chapter, we address the last research question RQ5 and present a context-aware scheme for SDVs to assess the risk of security incidents based on the vehicle's context, intending to identify adequate countermeasures automatically. Our scheme is inspired by the risk assessment process of the ISO/SAE 21434, which uses attack paths to model static threat scenarios. We specifically focus on attack propagation effects as a typical cyberattack is usually not a single event but a chain of events. Therefore, our scheme dynamically queries an Asset Dependency Graph (ADG) once a security incident is reported to identify attack paths leading to pre-assessed damage scenarios. Based on a risk value, the vehicle selects and realizes a compensating reaction.

This chapter is organized as follows: Section 8.1 presents the risk assessment process of the ISO/SAE 21434 standard. Section 8.4 describes the scheme we suggest for analyzing security incidents while the vehicle is in motion. We discuss the scheme in Section 8.5 and present a prototype implementation in Section 8.5.1. Our evaluation reveals that, on average, we require 0.61 ms to respond to a security incident in our setup.

This chapter builds upon research article "ISO/SAE 21434-based Risk Assessment of Security Incidents in Automated Road Vehicles" [7], which was presented at the *Computer Safety, Reliability, and Security* conference (SafeComp) in 2021. I thank Jonas Liske, who helped conduct the offline phase in Section 8.5. Students completed parts of the implementation and evaluation work as part of the Advanced Security Engineering Lab at the University of Passau.

8.1 The ISO/SAE 21434 standard

The ISO/SAE 21434 [35] is a cybersecurity engineering standard developed for road vehicles, and its first edition was published in 2021. This standard considers entire vehicle ecosystems by addressing organizational cybersecurity management, distributed cybersecurity activities, and the concept, development, and post-development phases. Comprising fifteen clauses, ISO/SAE 21434 provides a framework for common terminology, guidelines for managing security risks, cybersecurity policies, and processes. Specifically, the fifteenth clause introduces a Threat Analysis and Risk Assessment (TARA) process, which seems particularly suitable for our objective of assessing the propagating effects of security incidents, as it analyzes attack paths, a technique in modeling attack propagation.

The TARA process requires an item definition as mandatory input and involves nine consecutive compulsory steps, along with several optional ones. Organizations systematically identify threat scenarios and assess risks through this process to eventually determine appropriate defense techniques. Figure 8.1 visualizes the simplified methods presented in the fifteenth clause.

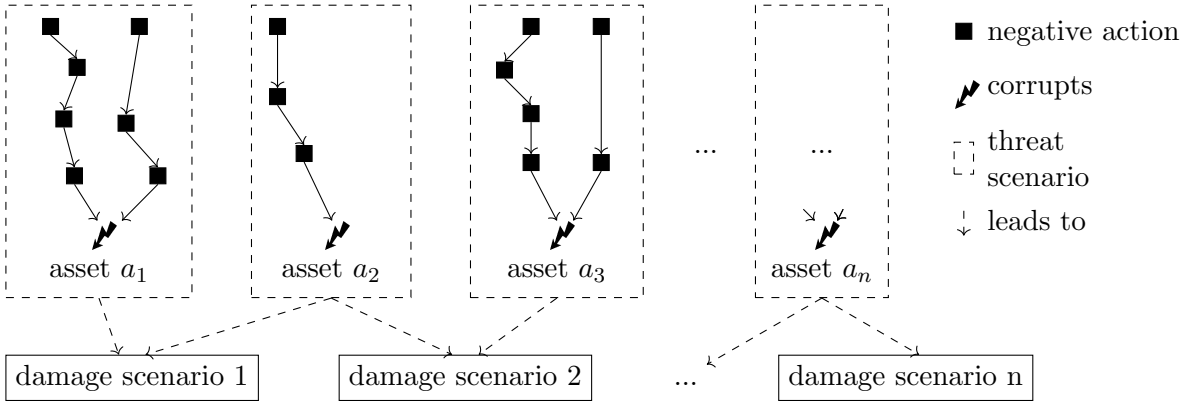

Figure 8.1: A threat scenario describes at least one series of negative actions (attack path) that leads to the corruption of an asset. This, in turn, causes at least one damage scenario. ISO/SAE 21434 computes the risk of threat scenarios by combining the attack path feasibility with the severity of the expected damage scenarios.

8.1.1 Risk Assessment Methods

The initial step of the TARA mandates the identification of damage scenarios, describing potential outcomes of cyberattacks and their relationships to item functionality, adverse consequences, and assets. Subsequently, critical assets whose compromise directly leads to a damage scenario must be identified. While we considered ECUs as assets during our security requirement analysis in Chapter 4, ISO/SAE 21434 adopts a broader definition, encompassing anything with the potential to cause a damage scenario, such as personal information or the transmission of critical commands.

The subsequent step involves the identification of threat scenarios, detailing how the corruption of critical assets may occur through multiple attack paths. A threat scenario specifies each asset's compromised cybersecurity properties and the cause of compromise. In the next step, the impact of each damage scenario is assessed in terms of safety, financial, operational, and privacy considerations, categorized as "severe", "major", "moderate", or "negligible". In addition, a safety-related impact rating is derived from ISO 26262, contributing to a security-safety co-engineering approach.

Once the severity of damage scenarios has been assessed, the threat scenarios are thoroughly analyzed by identifying attack paths within them. An attack path is a chain of dependent actions leading to asset corruption. For instance, the corruption of firmware may lead to the undetected injection of false messages to a physically shielded bus, eventually allowing $\mathcal{A}dv$ to control the vehicle. Once all threat scenarios are associated with at least one attack path, the attack feasibility for each path needs to be then determined. To achieve this, ISO/SAE 21434 proposes either an attack potential-based, a CVSS, or an attack vector-based approach. The resulting risk values, ranging from one to five, where one represents a minimal risk, are associated with threat scenarios.

In the final step, the organization selects appropriate risk treatment options for each threat scenario. Before this, it is essential to specify whether the risk should be avoided, reduced, shared, or retained.

8.1.2 Cybersecurity Assurance Level

ISO/SAE 21434 introduces the concept of Cybersecurity Assurance Level (CAL), a classification scheme to express assurance requirements for assets. The usage of CALs is comparable to the security levels used in the ISA-62443 standards, helping to derive technical requirements. While risks change over time depending on the attack surface and the deployed countermeasures, CALs remain constant after being determined typically in the concept phase. Similar to the security levels in ISA-62443 that either describe an entire zone or relate to individual foundational requirements, a CAL can be assigned to all cybersecurity goals or alternatively be used to describe individual goals. CALs are primarily used to determine methods for development, verification, the analysis of vulnerabilities, and security assessments.

Annex E of ISO/SAE 21434 provides insight into determining and using CALs. Instead of purely general guidelines, it provides examples for each level's notion and explains how impact and attack vector parameters can yield a specific CAL.

8.2 System Model

Researchers have presented numerous defense techniques for legacy and cutting-edge systems [116]. In this work, we expect vehicles to be equipped with state-of-the-art protection techniques meeting the security requirements identified in Section 4.3.3. These requirements include traffic and software integrity, intrusion detection, monitoring and logging mechanisms, and access control.

Specifically, we expect all benign ECUs to verify the authenticity of incoming data and discard unverifiable content. Software integrity checks are performed on vehicle startup, as explained in Chapter 7, preventing unauthorized, potentially malicious alteration to the software. Moreover, we assume that communication delays can be detected by ECUs [117] and that a firewall protects the in-vehicle network from external threats.

We expect any security violation to be reported to the central ECU $\mathcal{E}_{\mathrm{C}}$ through the in-vehicle network. It is essential to note that the transmission of security incidents introduces a new attack vector, as $\mathcal{A}dv$ might attempt to intercept security incidents, thereby preventing the vehicle from analyzing and reacting to them. Nevertheless, in this chapter, we assume uninterrupted reporting of security incidents, aware that further investigation is required to achieve this. That means that $\mathcal{E}_{\mathrm{C}}$ is always informed about suspicious and potentially malicious in-vehicle activities,

encompassing unverifiable traffic, illegal software changes, and the output of intrusion detection systems. $\mathcal{E}_C$ is considered a security anchor within the vehicle, similar to the ASP presented in Chapter 6. Consequently, $\mathcal{E}_C$ is considered trusted and will not be corrupted by $\mathcal{A}dv$. To achieve this, we refer to Section 6.2.2, discussing how to protect a trusted component technically. Regarding our reference vehicle, $\mathcal{E}_C$ can be implemented on the brainstem, which is designed to be fail-operational.

8.3 Security Incidents

ISO 27005 [118] defines a security incident as an event compromising a specific property, such as authenticity or availability. Such an event can result in physical damage and is initiated by a threat. In alignment with the ISO/SAE 21434 standard, we characterize a security incident as a threat scenario initiated by a negative event during vehicle runtime, which is then reported to $\mathcal{E}_C$. Thus, a security incident, perceived by a specific ECU, describes the violation of an asset's security property, with this property corresponding to a threat category derived from the deployed threat model.

We use the CIA triad, leading to $\mathsf{T} = \{\text{Confidentiality, Integrity, Availability}\}$, where T denotes the set of all threat categories. A threat category reflects the violated security property and is contingent on the applied threat model. While the ISO/SAE 21434 employs the CIA triad, other threat models such as STRIDE [119] or the foundational requirements used in ISA-62443 (c.f., Section 4.3.2) could offer more granular alternatives.

Negative events are mapped onto the relevant threat categories upon detection, as illustrated in Table 8.1. For instance, data originating from an unverifiable source and untrustworthy firmware affects "Integrity", while delayed safety-critical messages are associated with the category "Availability".

Negative Event	**Threat Category (CIA)**
Unknown Data Origin	Integrity
Unverifiable Firmware	Integrity
Unauthorized Data Access	Confidentiality
Delayed Traffic	Availability
...	...

Table 8.1: Negative events are mapped onto a threat category.

Let ECU be the set of ECUs and A denote the set of assets. We formally express a security incident ev as an element of $\mathsf{ECU} \times \mathsf{A} \times \mathsf{T}$. Hence, ev consists of the reporting ECU, the compromised asset, and the corresponding threat category linking the incident. For instance, the security incident ev =(*brainstem*, *door_open_cmd*, *Integrity*) states that the brainstem received a command to open the door whose integrity cannot be verified.

8.4 Context-Aware Reactions to Security Incidents

This section presents our context-aware assessment scheme to properly handle and react to security incidents in SDVs. We specifically focus on attack propagation effects since related works have demonstrated their relevance in automotive networks. We consider a security incident a

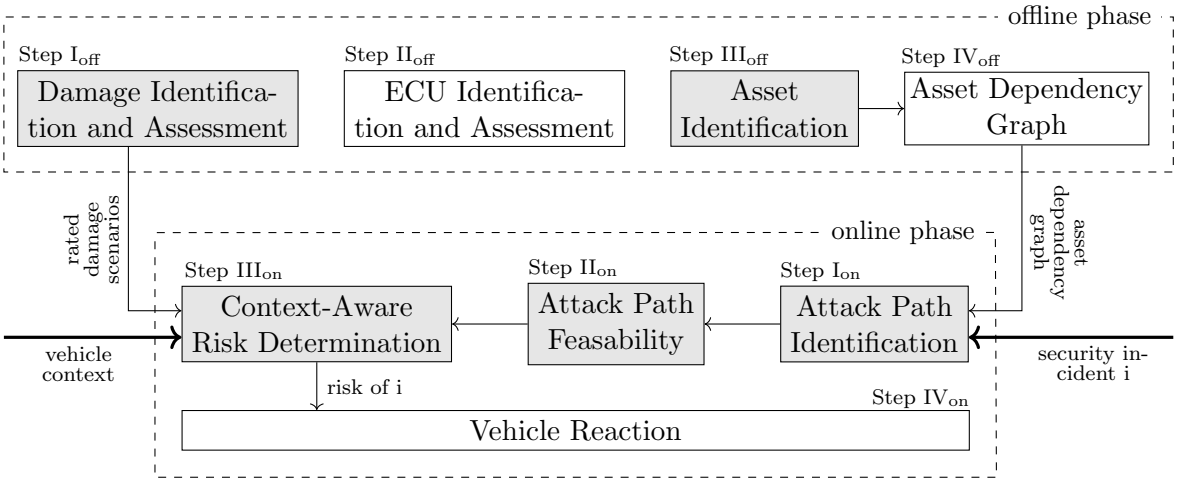

Figure 8.2: Structure of our context-aware risk assessment scheme for security incidents in automated road vehicles. White fields are not part of ISO/SAE 21434.

dynamic threat scenario and use an Asset Dependency Graph (ADG) to identify and rate possible attack paths. As shown in Figure 8.2, our scheme consists of an *offline* and an *online* phase. The offline phase identifies and rates damage scenarios, the vehicle topology, and, in particular, the ADG. They serve as input for the online phase, assessing the risk of a reported security incident and taking compensating action.

In the remaining sections, we explain each step of our scheme in detail, refer to the corresponding ISO/SAE 21434 requirements, and use the following notation: We denote $\mathcal{E}_C$ as the *monitoring* control unit that executes the online phase of our scheme. $\pi_k(x)$ returns the entry k of an ordered collection x (e.g., a tuple).

8.4.1 Offline Phase

The offline phase occurs once and is ideally carried out by experts during the vehicle design stage. This phase furnishes essential inputs for the subsequent online assessment phase, including rated damage scenarios and the ADG.

Step I$_{off}$: Damage Identification and Assessment

At first, we specify the set of all damage scenarios, denoted as D, in expert panel discussions. A damage scenario characterizes the anticipated outcome of security incidents, such as uncontrolled driving behavior. Experts collect, discuss, and rate damage scenarios in brainstorming sessions. In compliance with ISO/SAE 21434, each $d_i \in \mathsf{D}$ is ranked in the four categories: Safety, Financial, Operational, and Privacy (SFOP). For that purpose, we suggest implementing a multi-criteria decision-making process, such as the AHP method outlined in Section 4.3. This approach assigns varying weights to each impact category based on relevance. That way, a worst-case privacy violation is consistently regarded as less severe than a worst-case safety impact. Ultimately, the weighted scores are mapped to one of the four values: *negligible*, *moderate*, *major*, or *severe*. This mapping should align with the interpretation of those values provided in the ISO/SAE 21434 standard. For instance, a moderate impact on safety means light injuries, whereas a major impact indicates severe but non-fatal injuries.

Step $\mathbf{II_{off}}$: ECU Identification and Assessment

Subsequently, we identify ECU by enumerating all in-vehicle control units and assessing the attack potential AP for each $\mathcal{E}_\mathrm{i} \in \mathsf{ECU}$. ECUs hold a key role in our context-aware assessment scheme. Firstly, the propagation of attacks in road vehicles results from the communication between ECUs. Secondly, ECUs not only trigger damage scenarios but also detect and report security incidents. Hence, a compromised ECU poses a severe security threat as it can either directly cause damage or facilitate attack propagation, such as forwarding corrupted data. Therefore, it is crucial to include the attack potential AP of ECUs in the risk assessment of a security incident since a well-protected and resilient network of ECUs makes it harder for an attack to propagate. That means AP gives us a notion of how susceptible $\mathcal{E}_\mathrm{i}$ is to manipulation. As demonstrated in Section 3.1, cyberattacks on road vehicles often necessitate the manipulation of ECUs (e.g., to gain access to a physically shielded bus) for successful execution along a given attack path.

The rationale behind determining AP lies in the necessity of assessing the feasibility of attack paths in later steps during the online phase. Generally, we assume a higher path feasibility if the ECUs on that path have a larger attack potential, as it is easier for $\mathcal{A}dv$ to manipulate them. Note, however, that ISO/SAE 21434 does not prescribe to determine the attack potential. Instead, we introduced this step to facilitate the automatic assessment of the path feasibility. We describe each ECU $\mathcal{E}_\mathrm{i}$ as a tuple $\mathcal{E}_\mathrm{i} = (\mathrm{name}, AP)$ comprising a unique name and its attack potential AP.

To determine AP, we adhere to the ISO/IEC 15408 [120], also known as common criteria, ranking the minimum required attack resources in terms of *Elapsed Time*, *Expertise*, *Knowledge*, *Window of Opportunity*, and *Equipment*. Again, each parameter is associated with a distinct numerical value as indicated in Table 8.2. Eventually, we sum up these values to obtain a single attack score

Elapsed Time		**Expertise**		**Knowledge**		**Windows of Opportuniy**		**Equipment**	
Option	Value	Option	Value	Option	Value	Option	Value	Option	Value
<1 week	0	Layman	0	Public	0	Unlimited	0	Standard	0
<1 month	1	Proficient	3	Restricted	3	Easy	1	Specialized	4
<6 months	4	Expert	6	Confidential	7	Moderate	4	Bespoke	7
≤3 years	10	Multiple Experts	8	Strictly Confidential	11	Difficult	10	Multiple bespoke	9
>3 years	19								

Table 8.2: We employ Common Criteria methodology to determine the attack potential of ECUs.

for $\mathcal{E}_\mathrm{i}$, as suggested by ISO/SAE 18045 [121]. Subsequently, this value is uniformly mapped onto AP as shown in Table 8.3. We recommend performing this step as part of the post-development phase activities when the ECUs attack potential can be considered fixed.

Attack Score	0-9	10-13	14-19	20-24	>25
Attack Potential (AP)	0.9	0.7	0.5	0.3	0.1
Attack Feasibility	very high	high	medium	low	very low

Table 8.3: The attack score is mapped to an attack potential that indicates the attack feasibilty.

Step III$_{\text{off}}$: Asset Identification

In this step, we identify the assets A as required by ISO/SAE 21434. An asset describes anything whose compromise could lead to a damage scenario, either directly or through propagation effects. Consequently, assets are typically protected against manipulation, usually achieved through cryptographic measures and anomaly detection systems.

Our analysis of prevalent automotive attacks in Section 3.1 revealed that attackers frequently combine in-vehicle traffic manipulation with the intrusion of ECUs. In response, we distinguish between two types of assets: *flowing* and *rigid*. Flowing assets pertain to logically connected traffic between ECUs, such as services or messages on a specific topic. In contrast, a rigid asset resides on an ECU but can still impact flowing assets. Notably, these include software elements like services and firmware, as well as critical items such as cryptographic keys and configuration files.

Further differentiation is made between two subgroups, namely A_{t} and A_{p}, with $\mathsf{A} = A_{\text{t}} \cup A_{\text{p}}$ and $A_{\text{t}} \cap A_{\text{p}} = \emptyset$. A_{t} contains assets whose compromise directly triggers a damage scenario without any detour, meaning without attack propagation. For instance, the undetected corruption of the steering angle is likely to cause immediate harm. In contrast, A_{p} describes assets whose compromise leads to damage only through propagation effects. For example, a vulnerability in the infotainment firmware may enable an attacker to infiltrate forged commands, but it does not necessarily cause immediate harm.

Step IV$_{\text{off}}$: Asset Dependency Graph

Assets are interdependent because ECUs continuously communicate and perform computations on incoming assets. We write $a_y \leftarrow a_x$ to indicate that any change of asset a_x also influences asset a_y. This step aims to arrange the previously identified assets into an Asset Dependency Graph (ADG). This graph enables us to automatically check whether the corruption of a specific asset can transitively lead to damage. Formally, we describe the ADG as a directed multigraph $\text{ADG} = (\mathsf{V}, \mathsf{E})$. The set of vertices V comprises physical (V_p) and virtual (V_v) vertices, meaning that $\mathsf{V} = V_p \cap V_v$. ECUs are physical vertices with in-flowing and out-flowing assets. In contrast, we use a virtual vertex to indicate a dependency on a rigid asset residing on an ECU. For example, all outgoing assets of an ECU usually depend on its firmware.

We express an edge $e_i \in \mathsf{E}$ as a quintuplet according to Formula 8.1. That is, an edge $e_i \in \mathsf{E}$ exists between a source $\mathcal{E}_{\text{x}} \in \mathsf{ECU}$ and a target $\mathcal{E}_{\text{y}} \in \mathsf{ECU}$ if there is an asset $a \in \mathsf{A}$ flowing from $\mathcal{E}_{\text{x}}$ to $\mathcal{E}_{\text{y}}$. Each edge e_i is associated with a probabilistic weight w, indicating to what extent it contributes to a path in the ADG.

$$\mathsf{E} \subseteq \{(v_x, v_y, a, w, D) \mid (v_x, v_y) \in \mathsf{V}^2 \wedge a \in \mathsf{A} \wedge w \in [0,1] \wedge D \subseteq \mathsf{A} \setminus \{a\}\} \tag{8.1}$$

e_i also maintains a reference D containing those assets on which e_i *directly* depends on, i.e., $D = \{a_j \in A_{\text{p}} \mid a_i \leftarrow a_j \wedge \exists e_j \in \mathsf{E} \mid (\pi_a(e_j) = a_j \wedge \pi_{v_y}(e_j) = \pi_{v_x}(e_i))\}$. This is necessary because an out-flowing asset does not necessarily depend on all in-flowing assets. For instance, not every output of an ECU may be secured by a cryptographic key stored on that ECU.

The specification of ADG requires a profound knowledge of the network topology and the in-vehicle data flows, typically known only to the manufacturer. Creating such graphs is generally labor-intensive, so system designers may consider automated approaches [122].

8.4.2 Online Phase

The online phase is executed by $\mathcal{E}_\mathrm{C}$ every time a security incident is reported. In that case, $\mathcal{E}_\mathrm{C}$ identifies attack paths between the corrupted asset and any $a_i \in A_\mathrm{t}$ in ADG. Then, it determines their feasibilities and weights the severity of the expected damage with context information, resulting in a risk value of the security incident. Based on this risk, the SDV selects an appropriate reaction to the security incident.

Step I_{on}: Attack Path Identification

A security incident $\mathrm{i} \in \mathsf{ECU} \times \mathsf{A} \times \mathsf{T}$ reports the corrupted asset $a_x = \pi_a(\mathrm{i})$ on ECU $\mathcal{E}_\mathrm{x} = \pi_{ECU}(\mathrm{i})$. It requires further attention if it can lead to a damage scenario $d_i \in \mathsf{D}$, either directly or through propagation. This is the case if there is a path from $\mathcal{E}_\mathrm{x}$ to an $\mathcal{E}_\mathrm{y} \in \mathsf{ECU}$, whereas $\mathcal{E}_\mathrm{y}$ triggers d_i through the corruption of an $a_y \in A_\mathrm{t}$, which depends on a_x. We denote such a path $pth^{a_y \leftarrow a_x}_{\mathcal{E}_\mathrm{x}-\mathcal{E}_\mathrm{y}}$ and formally describe it as a series of edges in ADG as shown in Equation 8.2. For readability reasons, we later use pth_i to refer to a valid attack path in ADG.

$$pth^{a_y \leftarrow a_x}_{\mathcal{E}_\mathrm{x}-\mathcal{E}_\mathrm{y}} \in \{(e_1, ..., e_n) \mid e_i \in \mathsf{E}\} \tag{8.2}$$

The boundary conditions of Equation 8.2 are given by

$$\pi_{v_x}(e_1) = \mathcal{E}_\mathrm{x} \ \wedge \ \pi_{v_y}(e_n) = \mathcal{E}_\mathrm{y} \ \wedge \ a_y = \pi_{e_n}(a) \tag{8.2a}$$

$$\exists e \in \mathsf{E} \mid \pi_a(e) = a_x \ \wedge \ \pi_{v_y}(e) = \mathcal{E}_\mathrm{x} \tag{8.2b}$$

$$\forall e_{j,\, 2 \le j \le n} \mid \pi_{v_y}(e_{j-1}) = \pi_{v_x}(e_j) \tag{8.2c}$$

$$\forall e_{j,\, 2 \le j \le n} \mid \pi_a(e_{j-1}) \in \pi_D(e_j) \tag{8.2d}$$

That is, the path starts at $\mathcal{E}_\mathrm{x}$ and ends at $\mathcal{E}_\mathrm{y}$, where a_y may be corrupted through attack propagation (8.2a). Furthermore, there is an edge that leads into $\mathcal{E}_\mathrm{x}$ and contains the corrupted a_x (8.2b). All edges form a continuous path (8.2c) between ECUs. Besides, two adjoining edges have to carry dependent assets (8.2d), since otherwise attack propagation from a_x to a_y would not be possible. Note that $\mathcal{E}_\mathrm{C}$ may find multiple attack paths for a security incident leading to the same damage. Currently, we take the most feasible path among those with the worst impact, acknowledging that other strategies may also be reasonable.

Step $\mathrm{II}_{\mathrm{on}}$: Attack Path Feasibility

The ISO/SAE 21434 standard expresses the feasibility of each identified attack path as *high*, *medium*, *low*, *very low*. The feasibility indicates the likelihood of an attack being successfully carried out along a given path. As shown in Equation 8.3, we propose to calculate the feasibility $\mathcal{F}_{pth_i}$ of a path $pth_i = (e_1, ..., e_n)$ as the product of the corresponding edge weights w.

$$\mathcal{F}_{pth_i} = F_\mathrm{map}\Big(\prod_{j=1}^{n} \pi_w(e_j)\Big), \text{ with } F_\mathrm{map}(p) = \begin{cases} \text{high} & p \in [0.9, 1] \\ \text{medium} & p \in [0.5, 0.9[\\ \text{low} & p \in [0.2, 0.5[\\ \text{very low} & p \in [0, 0.2[\end{cases} \tag{8.3}$$

The edge weight is only determined in the online phase since it depends in particular on the security incident and the vehicle state. For instance, a DoS attack will likely propagate through large parts of the in-vehicle network as the communication slows down. In contrast, illegally

injected traffic only becomes harmful if ECUs process it instead of rejecting it. However, according to our system model, the latter only happens if the ECU is compromised because we assume a well-protected SDV. Thus, whenever a path along a specific edge requires the manipulation of an ECU, we take the previously determined attack potential AP from Step II_{off} as w. Besides, edges may be temporarily inactive, especially in a service-oriented environment (e.g., SOME/IP or ASOA). For instance, some services may only run when the vehicle is driving in a fully automated manner, while others are implemented for manual maneuvering. Inactive edges are assigned a zero weight, making the attack path infeasible. Furthermore, an SDV will likely possess multiple modes, e.g., for automated, manual, or remote maneuvering. Altogether, we distinguish between three cases for w:

1. $w = 0$: An edge weights zero if unavailable in the current vehicle state. For instance, a specific service is not running.
2. $w = 1$: An edge weight of one indicates a definite propagation between two vertices in ADG. This happens, for instance, if *Availability* is the threat category of an incident since delayed/dropped messages typically affect all subsequent assets. This also concerns rigid assets represented by virtual nodes. For example, the output of an ECU always depends on its firmware.
3. $w = \pi_{AP}(\pi_{v_x}(e_i))$: We use the attack potential AP of the edge's source $\mathcal{E}_{\text{x}}$ as edge weight if the manipulation of $\mathcal{E}_{\text{x}}$ is required for attack propagation, e.g., to circumvent security checks. This, for instance, may be necessary to transport forged traffic through the vehicle, as benign Electronic Control Unit (ECU) would discard it.

Step III_{on}: Context-Aware Risk Determination

Eventually, the risk of the security incident is determined by combining the attack path feasibilities with the expected severity of the associated damage scenarios. Recall that by definition (cf. Step $8.4.2_{\text{on}}$), an attack path always leads to exactly one damage scenario.

If worst-case damage were identified in Step I_{off}, we would probably obtain a high risk unless the attack path feasibility is "very low". However, assuming a fixed damage may not necessarily aid in pinpointing an appropriate reaction. For instance, considering the worst-case damage of a corrupted headlight control command, which could imply fatalities, an emergency stop seems reasonable during a night ride. Yet, during daylight hours, an emergency stop might be excessive, and driving at a reduced speed may be a more acceptable alternative (e.g., to reach the nearest repair shop). We suggest weighing the damage with vehicle context parameters for a realistic understanding of the expected damage.

Note that we are not the first to use such parameters for modeling the vehicle context, although related works typically do this in different fields. For instance, Helmholz et al. [123] consider the daytime and the route frequency for predicting trajectories. Since the latter is extraneous for the instant assessment of expected damage, we use the current speed and the traffic density instead. We express the vehicle context as a vector $\vec{C} = (\text{S TD T RQ})^{\intercal}$, consisting of the four parameters *Speed* (S), *Traffic Density* (TD), *Time* (T), and *Route Quality* (RQ).

We allow only Boolean values for each parameter to maintain simplicity, as illustrated in Table 8.4. For example, we distinguish between low speed (<30km/h) and high speed (≥30km/h), day and night drive, and so forth. Recognizing that these criteria do not all carry the same weight in influencing the severity of a damage scenario, we assign weights to $\vec{C}$ using the normalized

	Speed (V)	Traffic Density (TD)	Time (T)	Route Quality (RQ)
Weight	0.5	0.3	0.1	0.1
Value	low (0.5) high (1)	low (0.5) high (1)	day (0.5) night (1)	easy (0.5) difficult (1)

Table 8.4: Criteria for assessing the vehicle context

vector $\vec{W}_{\mathrm{C}} = (0.5\ 0.3\ 0.1\ 0.1)^{\intercal}$. Consequently, vehicle speed contributes five times more to the context than the route quality. The expert panel determined these weights, which are subject to future adjustments if necessary.

We calculate a scalar representation of the vehicle context by computing $C = \vec{C} \cdot \vec{W}_{\mathrm{C}},\ C \in [0.5, 1]$. This scalar value is then multiplied by the numerical damage assessment of Step $\mathrm{I}_{\mathrm{off}}$, aiming to achieve a more realistic, context-aware assessment of the expected damage.

Finally, we obtain the context-aware risk r_{i} for the security incident i using the risk matrix in Table 8.5, as proposed in ISO/SAE 21434.

		Attack Path Feasibility $\mathcal{F}_{pth_i}$			
	Risk r_{i}	**very low**	**low**	**medium**	**high**
Impact of damage scenario	**negligible**	1	1	1	1
	moderate	1	2	2	2
	major	1	2	3	4
	severe	1	3	4	5

Table 8.5: Risk matrix from ISO/SAE 21434 used to assess the risk of a security incident

Step $\mathrm{IV}_{\mathrm{on}}$: Vehicle Reaction

Finally, the risk value r_{i} is translated into an appropriate compensating vehicle action. We identified four compensating actions (cf., Table 8.6) in brainstorming sessions. An emergency stop is mandated when the anticipated outcomes involve fatalities, severe injuries, or substantial financial losses. Conversely, driving at reduced speed is considered appropriate if the security incident is expected to result in operational limitations (e.g., traffic jams) or minor injuries. We argue that this option is similar to the run-flat system of contemporary vehicles, which are activated in case of moderate damage.

If damage to the vehicle and passengers is expected to be only feasible with considerable effort, the vehicle displays a dashboard control message. This option is also selected in cases involving

Risk Value	**Compensating Action**
1	log/report incident + continue driving without restrictions
2	log/report incident + display a dashboard control message
3	log/report incident + driving at low speed
4, 5	log/report incident + emergency stop

Table 8.6: The context-aware risk of a reported security incident leads to a vehicle reaction.

potential privacy violations. Lastly, for incidents resulting in negligible damage, the response is limited to logging and reporting the event to an authority without additional actions.

8.5 Evaluation

In this section, we implement and evaluate the proposed scheme, taking two aspects into account:

Firstly, we are interested in the response time to security incidents. Given that we intend to react on-the-fly to potential security breaches, the vehicle's response time should be as small as possible. More precisely, we investigate whether an SDV can react faster than a human passenger could do.

Secondly, we explore the quality of the vehicle reaction, indicating whether it aligns with the perceived incident. One motivation behind the assessment of security incidents is to keep SDVs operable as long as possible, thereby preventing an emergency halt each time an anomaly is detected.

As part of our evaluation, we construct a Asset Dependency Graph (ADG) for our reference vehicle presented in Section 4.1. Additionally, we generate artificial graphs to investigate the scheme's behavior as the graph expands in size.

8.5.1 Setup

Initially, we implemented the scheme in C++ and deployed it as a standalone binary on the application processor of the brainstem. As detailed in Section 4.1.1, the brainstem is a central component in the reference vehicle designed to be fail-operational and consists of a real-time and application processor. After having added the binary to the autostarts, the brainstem listens on a fixed port for incoming security incidents.

Figure 8.3 visually represents our physical setup, involving three hardware components: two representing conventional ECUs, and the third representing the brainstem. We created two dummy ASOA services, A and B, each consisting of a guarantee and a requirement, and deployed them on the ordinary ECUs. Service A generates a random integer and publishes it through its

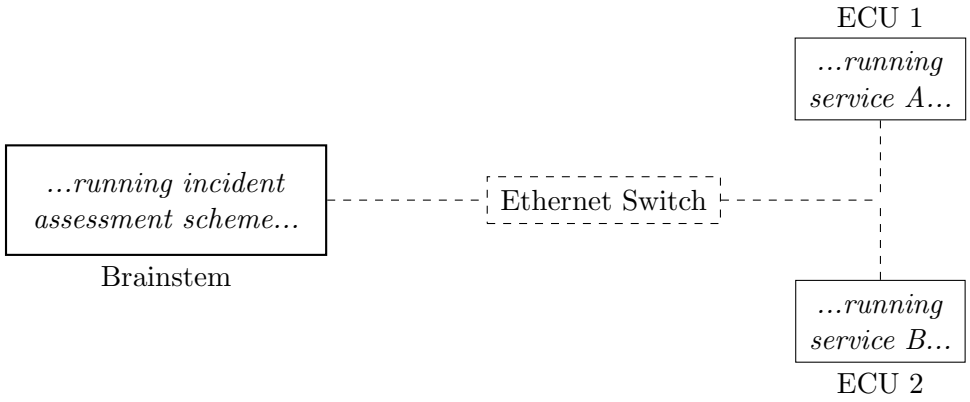

Figure 8.3: Our evaluation setup comprises two dummy ECUs, exchanging authenticated integers. ECU 1 intentionally transmits a false MAC every 500 iterations, triggering a security incident on ECU 2. Subsequently, ECU 2 reports the incident to the brainstem, where the prototype implementation of our assessment scheme is running.

guarantee, while service B receives this integer, verifies its authenticity, adds one, and publishes it again. This creates a ping-pong game where the transmitted number increments by one in each iteration. Every 500 iterations, service A deliberately transmits a wrong MAC, triggering a security incident on the counterpart that is instantly reported to the brainstem.

Both services run on dedicated hardware, namely two RPis, which are also deployed in our reference vehicle. Although they do not comply with automotive requirements, they facilitate prototyping. Whenever a false MAC is detected, the security incident is reported to the brainstem, which runs the proposed assessment scheme. Similar to the reference vehicle, communication between the dummy ECUs and the brainstem occurs through a switch connected to the same Ethernet network.

Damage Scenarios and ECU Identification

As required by the proposed scheme, we first conducted the offline phase, starting with identifying and rating damage scenarios in Step $\mathrm{I}_{\mathrm{off}}$. In total, we identified 8 damage scenarios. Each was rated according to the expected worst-case impact across the four SFOP categories, as presented in Table 8.7. The first damage scenario, denoted as d_1 = *Uncontrolled Driving*,

ID	Damage Scenario	S	F	O	P	Weighted Score	Result
d_1	Uncontrolled Driving	0.5	0.5	0.5	0	0.438	Severe
d_2	Manipulated Vehicle Routing	0.167	0.167	0.5	0	0.087	Major
d_3	Passenger Inconvenience	0.167	0	0.333	0	0.134	Moderate
...	...		...			...	...
d_8	Battery Degradation	0	0.5	0.333	0	0.078	Major

Table 8.7: In total, eight damage scenarios have been identified and rated with regard to their worst-case impact on safety, financial, operational, and privacy.

describes a malicious party's illegal takeover of the dynamic modules and is probably the most feared consequence of an attack. In the worst case, not only fatalities are likely to occur, but also severe financial and operational impacts because of legal consequences such as lawsuits. In contrast, d_3 =*Passenger Inconvenience* describes limitations arising mainly from manipulations within the vehicle interior, such as a tampered heating system or blocked doors. Instead of severe physical harm, we expect financial damage due to patching and a loss of reputation.

Moving on to Step $\mathrm{II}_{\mathrm{off}}$, we identified 26 ECUs and determined their attack potential AP. For instance, the well-protected brainstem yielded an absolute attack score of 29, corresponding to the low attack potential $AP = 0.1$ (refer to Table 8.3). In contrast, the HMI exhibited a high attack potential of $AP = 0.9$ primarily attributed to external interfaces and user input.

Asset Dependency Graphs

For a realistic evaluation, we construct the full ADG for our reference vehicle by systematically querying the architecture tool introduced in Section 6.1. Recall that this web-based tool is connected to a database describing the vehicle's hardware and software architecture, including communication links between ASOA services. For simplicity, we make the following assumptions while creating the ADG:

- We exclusively consider communication between ASOA endpoints, meaning that flowing assets are exchanged between guarantees and requirements.
- We add a rigid asset to each ECU, representing its firmware or OS, respectively.
- All outgoing guarantees depend on all incoming requirements.

In particular, the last assumption needs adjustment, particularly when deploying our scheme in a road vehicle, as not every incoming data affects all outgoing computations. This is especially not the case in a service-oriented and zonal architecture where fewer ECUs perform multiple and independent tasks. Our assumption leads to more asset dependencies and, consequently, to an increase in attack paths, possibly making the overall assessment oversensitive. However, it simplifies the generation of the ADG and is expected to produce rather conservative evaluation numbers since more attack paths must be considered.

In total, we identified 32 assets in Step III_{off}, whose corruption can directly cause at least one of the previously identified damage scenarios, i.e., $|A_{\text{t}}| = 32$. For instance, $d_1 =$ *Uncontrolled Driving* can result from the corruption of the assets $a_1 =$"Steering Angle", $a_2 =$"Torque", $a_3 =$"Rotational Frequency", and/or $a_4 =$"Firmware Dynamic Module". After that, we identified 82 assets from A_{p} in a bottom-up approach, focusing on assets capable of causing damage through propagation. In summary, our ADG comprises 173 edges and 59 nodes. Note that the number of nodes exceeds the number of ECUs because we introduced rigid nodes for firmware assets that do not flow between ECUs but still impact other assets (see Step IV_{off}). The fan-in and fan-out of each node indicate that most dependencies exist between the brainstem, the cerebrum, and the sensor modules. This finding is unsurprising, given that the main event chain leads along these ECUs. For instance, the brainstem has a fan-out of 26, while smaller ECUs have an average fan-out of only three.

Eventually, we deploy the complete ADG as an adjacency list on the brainstem. Every time a security incident is reported by service B, we randomly map the service on one of the ECUs of the prototype vehicle. This approach introduces slight variations in each incident compared to the previous one, enhancing the meaningfulness of our measurements.

Synthetic Graphs Besides the ADG taken from the reference vehicle, we repeat our evaluation experiment with larger, artificially crafted ADGs. That way, we aim to assess the impact of the graph size, which we expect to increase in a production vehicle. Along with the number of nodes, the graph complexity is a crucial parameter of artificially created ADGs. The complexity denotes the maximum number of connections each node can have. A higher complexity can eventually lead to a mesh network, thereby shortening attack paths.

Results

Table 8.8 summarizes the results of our experiments. We compared three well-known algorithms for (attack) path identification: Dijkstra, a breadth-first search, and Bellman-Ford. For each measurement, we computed the mean, minimum, and maximum time. As anticipated, the Dijkstra implementation outperforms the others, while a complete traversal of the ADG proves to be the slowest. Our benchmark is to surpass human reaction time, typically around 250 ms.

Using a Dijkstra implementation with the reference vehicle's ADG, it took an average of 0.61ms to detect, report, and assess a security incident. In contrast, the assessment time increases to 0.68 ms and 0.71 ms when utilizing Bellman-Ford or a breadth-first search for attack path

	Dijkstra (ms)			**Bellman-Ford** (ms)			**Breadth first** (ms)		
#Nodes	59	1000	10000	59	1000	10000	59	1000	10000
Mean	0.61	68.29	3,163.34	0.68	41.48	3,224.13	0.71	43.78	3,385.811
Min	0.41	0.46	0.614	0.417	0.54	0.69	0.51	0.54	0.608
Max	0.76	71.09	4,284.27	0.58	58.87	3,985.27	0.89	51.31	3,866.18

Table 8.8: We measured the time it takes to report and assess a security incident using three different algorithms.

identification. We find these values acceptable since they lie below our baseline of 250 ms, meaning a vehicle can opt for an appropriate safety measure much faster than a human.

In contrast, we observe a substantial increase as the number of edges and nodes grows. For instance, with an artificial graph containing 10,000 nodes and a complexity of 10, the average response time exceeds three seconds for all deployed algorithms, far beyond an acceptable range. Hence, maintaining a slim ADG is crucial because otherwise, a safety reaction may be initiated too late. This involves reducing dependencies between in-vehicle components, both hardware and software, to generate smaller ADGs. Non-functional dependencies can be reduced through segmentation, a requirement that we had already identified in Section 4.3.3 as a result of our security requirement analysis.

Our analysis shows that assessing security incidents is feasible during vehicle operation if dependencies are kept low and the ADG as light as possible.

Optimizations

To further optimize the time complexity, it is possible to enhance the implementation of the attack path identification algorithm. Internally, we store the ADG as an adjacency list, resulting in a time complexity of $\mathcal{O}(|V|^2)$ with V indicating the set of nodes. However, we could achieve $\mathcal{O}(|V|\log(|V|) + |E|)$ by utilizing a Fibonacci heap [124] in our Dijkstra implementation. This improvement results in a more efficient algorithm execution, particularly beneficial as the size of the ADG grows.

8.5.2 Discussion

In the previous section, we showed that a sufficiently short response time is feasible. However, a short response time is only one side of the coin. The other side is concerned about the quality of the selected reaction. Recall that we want to keep the vehicle operating as long as possible instead of always initiating an emergency stop. Given the impracticality of observing our scheme's reactions in real traffic over an extended period, we simulate security incidents and discuss the compensating action in this chapter. Before that, we discuss the general efficacy of attack assessment and whether our scheme is a solution to making vehicles more resilient to cyberattacks.

Attack Detection

The online phase assesses the risk of security incidents, prompting the question of which attacks can be uncovered. Given that a security incident follows a negative event, our scheme can only detect and address attacks that the deployed security mechanisms can perceive. Therefore, a

thorough security requirement analysis, as we did in Chapter 4, is a crucial prerequisite during the vehicle design phase, as it is only then that necessary defense techniques can be identified and subsequently realized.

We categorize each negative event into a threat category, thereby simplifying the determination and assessment of attack paths. However, this approach comes with a trade-off - while it streamlines the process, it results in a reduced attack resolution, with different attacks mapped to the same threat category. Currently, we employ the CIA triad and allocate all security incidents to one of these three categories. To mitigate the resulting loss of information, a more fine-grained threat model, such as STRIDE [119] or the foundational requirements of ISA-62443 (see Section 4.3.2) can be considered.

Furthermore, some negative events may not be uniquely associated with one threat category. For instance, "unauthorized data access" can affect both *Integrity* and *Confidentiality*. In such instances, multiple security incidents could be reported for a single negative event and processed by $\mathcal{E}_{\mathrm{C}}$ based on their respective risk levels.

Ultimately, automotive organizations must decide the desired trade-off between attack resolution and model complexity.

Compensating Actions

Our scheme selects a compensating action based on the computed risk of the security incident. In this way, we aim to protect the passenger from dangerous cyberattacks while ensuring the continued operability of the vehicle for as long as possible. The severity of a security incident depends on the attack path feasibility and the expected damage, with the latter considering the vehicle context.

To simulate security incidents, we assumed corruption of the assets in A_{t}. Subsequently, we investigated how frequently each compensating action is taken under fixed worst-case damage conditions (i.e., we assume the most unfavorable vehicle context) but for varying attack path feasibilities. Figure 8.4 illustrates that smaller feasibility typically leads to a weaker compensating action because more attacker capabilities are required to propagate an attack successfully. More precisely, an emergency stop is necessary for 30% of security incidents with expected high

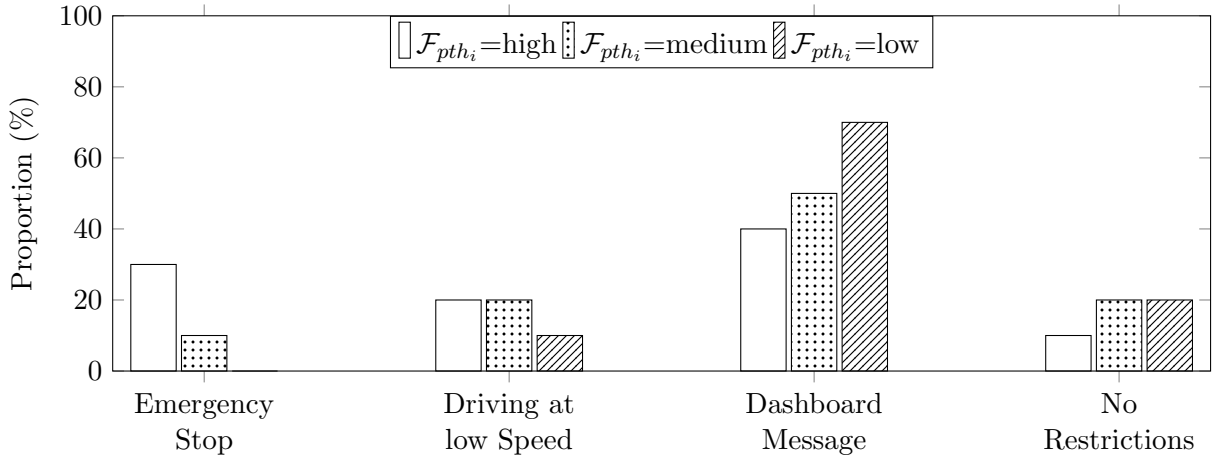

Figure 8.4: Assuming a constant worst-case damage, a smaller attack path feasibility usually leads to a weaker compensating action.

feasibility. In contrast, no emergency stop is triggered when the feasibility is low; weaker actions like dashboard messages are selected (up to 70%). We note that this distribution does not fully reflect reality, as not all assets in A_t are usually compromised with equal frequency. However, it confirms our intention for an adequate compensating action - treating non-critical security incidents less severely than those jeopardizing the passenger's well-being.

The application of our scheme can help interpret an SDV's security state but requires awareness for attacks that take effect immediately and are difficult to assess. For instance, in case an attacker gains access to a safety-critical ECU and can immediately corrupt a critical asset from 32, our scheme is unlikely to prevent this from happening. However, attacks with a long attack path or those that gradually evolve over time, such as the flooding of an in-vehicle network, the compromise of a seemingly uncritical component, or access violations, can be handled and mitigated.

8.6 Sub-Conclusion

This chapter dealt with the question of how a vehicle should respond to a security incident. To answer this question, we introduced a scheme allowing us to assess security incidents concerning the expected damage. This scheme intends to enhance the resilience of SDV to security incidents occurring during vehicle operation. It is the last building block of our holistic security engineering approach, as it covers the operation phase in which it enables SDVs to select a compensating action for a previously reported incident automatically. The objective is to prevent damage and passenger harm while keeping the SDV operable as long as possible.

We specifically focussed on attack propagation, as popular attacks on road vehicles have illustrated how manipulating even minor ECUs may allow attackers to infiltrate substantial vehicle components. Our work is based on the novel automotive cybersecurity standard ISO/SAE 21434 that utilizes attack paths to model in-vehicle dependencies for the risk assessment of threat scenarios. This concept is transferred to vehicle operation time by treating a security incident as a dynamic threat scenario. The core idea involves identifying attack paths within an asset dependency graph and determining their feasibility with respect to the security incident. The offline phase of our scheme was applied to our reference vehicle, for which we crafted an asset dependency graph consisting of 59 nodes and 173 edges.

Our evaluation demonstrates that detecting, reporting, and assessing security incidents is possible below the human reaction time. We observed an average response time of 0.61 ms using an ADG representing our reference vehicle. However, it was also apparent that the response time highly depends on the ADG's size and algorithm for identifying attack paths. Keeping asset dependencies low in SDV results in fewer attack paths, hence contributing to the SDV's security. This can be achieved by clearly specifying which input of a service impacts which output.

Finally, we attempted to anticipate the quality of the selected vehicle reaction, demonstrating that high-risk incidents typically lead to an emergency stop, while less severe incidents trigger weaker actions. Moreover, we pointed out it is important to remember that some cyberattacks take effect instantaneously (e.g., a software exploit), while others evolve, particularly those on the network, spoofing, and data leakage.

Considering these factors, we consider our scheme a step towards a resilient SDV and an answer to our last research question RQ5.

9. Conclusion

The transformation of road vehicles into dynamic, interconnected, software-driven systems has significantly expanded the attack surface for potential threats. Numerous cyberattacks have demonstrated how adversaries can compromise vehicles and even take control of entire vehicle fleets, posing a substantial risk to passenger safety. Since then, there has been a growing demand for enhanced security within the automotive industry, as a safe vehicle must be resilient to cyberattacks.

In this thesis, we followed a holistic approach to develop security concepts that address various stages of a vehicle's lifetime. Several of our proposed concepts were integrated and evaluated in the four automated, software-driven UNICARagil vehicles, resulting from a national research project funded by the German government. Our work was guided by five central research questions (referenced as RQ1 - RQ5), each tackling a specific security aspect along the vehicle's lifecycle.

At first, we conducted a risk analysis of the software and hardware architecture of the UNICARagil vehicles, aligning with the ISA-62443 industry standards. This analysis led to the derivation of security requirements aimed at reducing the overall cybersecurity risk of previously identified threats to a tolerable level. Our analysis revealed system integrity, access control, network and application segmentation, DoS protection, monitoring of, and the ability to react to security breaches as top-priority requirements, directly addressing RQ1. While several technical solutions have already existed for these requirements, the key for SDVs lies in having adaptable and maintainable solutions, given an average vehicle lifetime of twenty years. Besides defense techniques, the answer to RQ1 also encompasses the need for an overall security process capable of quickly reacting to new security breaches without necessitating costly system interventions.

Subsequently, we addressed selected security requirements, particularly system integrity, as popular cyberattacks exploited missing integrity features. We started investigating the security of the CAN and FlexRay legacy protocol, proposing efficient means to compute and distribute cryptographic keys while ensuring the authenticity of in-vehicle traffic. In response to RQ2, our findings emphasize the importance of authenticity for safety-critical traffic within these networks. Achieving communication security involves cryptographic protection and implementing a systemwide configuration and maintenance process, enabling the computation and distribution of cryptographic keys to both resource-constrained and high-performance control units.

Such a process becomes even more critical for automotive middleware stacks as they support the dynamic loading and relocation of software components alongside service-oriented communication subject to regular updates and changes. Addressing RQ3, we introduced a three-stage process. This process centralizes the administration of a vehicle's security specification, computation, and secure distribution of security tokens according to that specification. Additionally,

it involves the enforcement of desired security objectives during runtime. A widespread consortium of researchers has used this process during the software integration of the UNICARagil vehicles.

To ensure the integrity of automotive software within SDVs, we expanded the vehicle's bootup process with an integrity measurement scheme using remote attestation techniques. Like mandatory functional and operational tests of contemporary automobiles, we envisioned a security check once an SDV intends to participate in (automated) traffic. To answer RQ4, we suggest a scheme where a legal authority verifies and validates a previously computed integrity identifier and, upon successful verification, issues a certificate. That certificate serves as proof of the vehicle's trustworthiness to other parties, providing a mechanism to prevent malicious vehicles from accessing the roads.

Finally, we shifted our focus to the runtime behavior of a security-aware SDV, which should effectively respond to security incidents. Such incidents can be triggered by false alarms or actual attacks. In response to RQ5, we introduced a context-aware assessment scheme determining the worst-case impact a security incident could cause. This involves specifying and evaluating attack paths within an asset dependency graph, which, among others, considers in-vehicle data flows, ultimately leading to the implementation of appropriate safety measures. We implemented our scheme and demonstrated that detecting, reporting, and assessing security incidents in our reference vehicle is possible below the human reaction time. We pointed out that the response time depends on the number of dependencies within the SDV and the algorithm used to identify attack paths on the fly. While our scheme does not prevent cyberattacks from happening, it contributes to the resilience of SDVs.

In summary, this work has investigated the systematic analysis of security requirements for automobiles, the design of both reactive and proactive protective measures to defend against cyberattacks, and how vehicles should respond to security alarms. Our solutions for these aspects have resulted in holistic security engineering covering different phases of a vehicle's lifecycle. While security is a relatively new aspect in the field of automotive engineering, it has become a must-have for vehicles, especially as they transition into software-controlled, remotely updatable, and interconnected entities. Yet, effective protective measures are only one side of the coin, mainly because, in most cases, the cryptographic tools for securing automotive systems already exist. The other side encompasses the need for a security management process to prepare these protective measures for real-world applications, demanding long-term maintenance support. Precisely, this process must reflect the adaptability and maintainability properties of the underlying software architecture and align with legal and organizational demands. We conclude that the combination of effective protective and monitoring measures, along with such a process, is essential for ensuring the security and, ultimately, the safety of SDVs.

9.1 Outlook

The transformation of road vehicles into software-defined systems affects technological developments and influences the entire automotive market, as explained at the beginning of this dissertation. We anticipate drastic changes in the market due to new participants from around the globe, all trying to catch on to developments towards electrified and automated traffic. The

traditional market, as found in Germany, will have to adjust, as we can already see with OEMs taking again direct responsibility for software architectures.

Another potential area of growth in the automotive domain lies in the rapid advancements in artificial intelligence. A perhaps absurd idea is a huge neural network controlling the entire vehicle such that all current efforts in redesigning vehicular hardware and software architectures become superfluous. Whether this idea is only a dream or can become true will show the future.

In the short term, further developments are likely to occur in the following four points, particularly concerning the field of automotive security:

Security-by-Design The digitalization and automation of road vehicles are ongoing processes, so designing secure automobiles remains a challenge with open research questions. In particular, investigating how the automotive industry can adopt a security-by-design process is essential, mainly due to the multitude of in-vehicle software and hardware components provided by suppliers. Therefore, a proper security-by-design approach requires full stakeholder collaboration to establish secure development practices along the supply chain. Automotive industry regulations specifying standard compliance rules are necessary to facilitate these collaborative efforts and enable a holistic and proactive security engineering approach. All stakeholders must recognize that security is a foundational element of vehicle design and development. In 2021, a promising initial step toward that direction was undertaken with the publication of the ISO/SAE 21434 standard. Nevertheless, whether in discussions with industry professionals or research colleagues, we often encountered the prevailing attitude of prioritizing security in the aftermath rather than integrating it into significant design decisions. A security-by-design approach helps the industry systematically address cybersecurity threats and resulting attacks, provides evidence for regulatory authorities, and offers liability protection. In this context, governmental authorities must specify compliance and regulatory requirements in greater detail for SDVs to meet at a minimum level.

Configurable Security Another trend we anticipate describes configurable security solutions, particularly beneficial in service-oriented environments where software can be dynamically loaded and remotely updated. Taking communication security as an example, fundamental properties like traffic authenticity and confidentiality could be achieved using technologies such as IEEE 802.1AE, which operates at the link layer. Despite the clear performance benefits, such solutions can only be configured coarse-grained and require significant management efforts, particularly in dynamic systems. At the link layer, distinct data flows become invisible, and additional security requirements like access control are challenging to implement. While we acknowledge the importance of security solutions being both performant and transparent to applications, we also believe they need to be as configurable and adaptable as the overall middleware.

Fine-grained security configuration and the potential for adaptability require support from the deployed middleware, also known as automotive OS. Currently, it is still unclear which automotive OS will prevail. Although many automotive companies affirm the value of a common solution, many are still working on individual products, as pointed out in Section 1.2. In the coming years, either the market will converge toward a specific or a limited number of automotive OS options, or a variety of solutions will coexist. OEM companies increasingly exert control over the software development branch, which has been outsourced to suppliers for many years. The direction in which automotive software architectures evolve will determine the extent of the shift in software responsibility.

Automotive Hardware Not only are automotive software architectures evolving, but the underlying hardware is also undergoing a fundamental transformation. We expect the trend towards centralized and zonal E/E architecture to continue gaining speed while hardware and software are becoming increasingly decoupled. Modern will ECUs likely consist of multiple microprocessors, typically an application and a real-time processor, as is the case with the UNICARagil brainstem. For maximum protection against adversaries, automotive hardware must inherently provide security features, including secure storage for cryptographic secrets and long-term maintenance of identities. Furthermore, hardware-accelerated implementations of cryptographic primitives can enhance overall system performance, while secure environments facilitate the execution of critical code. For example, ARM processors come equipped with such security features. They are renowned for their energy efficiency, support hardware virtualization, and are optimized for running safety-critical code that must adhere to standards like ISO 26262. These characteristics make them suitable for automotive applications. Beyond technical aspects, we anticipate greater modularity in automotive hardware, similar to the ongoing transformations in the software layer. Modular hardware systems are easier to maintain and improve cost-efficiency, which are essential goals in the competitive automotive industry.

Security Measures The design and maintenance of secure vehicle systems unquestionably require a security-by-design approach and robust management processes. However, the actual protection of SDVs against adversaries ultimately relies on effective and efficient defense measures. While we have presented solutions for only a selection of the identified security requirements, SDVs must address all requirements. Beyond the measures presented in this thesis, a secure SDVs necessitates privacy-friendly technologies, especially when operating autonomously and utilizing sensors for environment perception. While our work primarily relied on cryptographic solutions to detect adversaries, intrusion detection systems offer an alternative approach to detecting adversaries. Although analyzing anomalies in computer systems is not new, its applicability in vehicular contexts requires further exploration, particularly regarding real-time behavior and reliability. In addition, the virtualization of automotive software, the physical or virtual separation of critical and non-critical traffic, and the development of quantum-resistant cryptographic primitives are also significant considerations in the design of secure SDVs.

We argue that technical solutions exist for most security challenges, making entirely new fundamental security research optional. However, the adaptation and applicability of these solutions in the context of vehicles necessitate thorough investigation, taking into account safety, timing, and performance requirements while adhering to industry standards.

In summary, as soon as security is no longer considered an annoying feature but an integral part of the vehicle development and maintenance process, the vision of secure, safe, highly automated self-driving vehicles is within reach.

Bibliography

[1] Dominik Püllen, Nikolaos Athanasios Anagnostopoulos, et al. "Safety Meets Security: Using IEC 62443 for a Highly Automated Road Vehicle". In: *Computer Safety, Reliability, and Security*. 2020, pp. 325–340. DOI: 10.1007/978-3-030-54549-9_22.

[2] Dominik Püllen, Nikolaos Athanasios Anagnostopoulos, et al. "Using Implicit Certification to Efficiently Establish Authenticated Group Keys for In-Vehicle Networks". In: *2019 IEEE Vehicular Networking Conference (VNC)*. 2019. DOI: 10.1109/VNC48660.2019.9062785.

[3] Dominik Püllen, Nikolaos Athanasios Anagnostopoulos, et al. "Securing FlexRay-based in-vehicle networks". In: *Microprocessors and Microsystems* 77 (2020), p. 103144. DOI: 10.1016/j.micpro.2020.103144.

[4] Dominik Püllen, Florian Frank, et al. "A Security Process for the Automotive Service-Oriented Software Architecture". In: *IEEE Transactions on Vehicular Technology* (2023). DOI: 10.1109/TVT.2023.3333397.

[5] Dominik Püllen, Nikolaos Athanasios Anagnostopoulos, et al. "Poster: Hierarchical Integrity Checking in Heterogeneous Vehicular Networks". In: *2018 IEEE Vehicular Networking Conference (VNC)*. 2018. DOI: 10.1109/VNC.2018.8628375.

[6] Dominik Püllen, Felix Klement, et al. "Ensuring Trustworthy Automated Road Vehicles: A Software Integrity Validation Approach". In: *2023 IEEE International Automated Vehicle Validation Conference (IAVVC)*. 2023. DOI: 10.1109/IAVVC57316.2023.10328103.

[7] Dominik Püllen, Jonas Liske, and Stefan Katzenbeisser. "ISO/SAE 21434-Based Risk Assessment of Security Incidents in Automated Road Vehicles". In: *Computer Safety, Reliability, and Security*. 2021, pp. 82–97. DOI: 10.1007/978-3-030-83903-1_6.

[8] Florian Kohnhäuser, Dominik Püllen, and Stefan Katzenbeisser. "Ensuring the Safe and Secure Operation of Electronic Control Units in Road Vehicles". In: *2019 IEEE Security and Privacy Workshops (SPW)*. 2019, pp. 126–131. DOI: 10.1109/SPW.2019.00032.

[9] Timo Woopen, Bastian Lampe, et al. "UNICARagil - Disruptive Modular Architectures for Agile, Automated Vehicle Concepts". In: 27. Aachen Colloquium Automobile and Engine Technology 2018. 2018, pp. 663–694. DOI: 10.18154/RWTH-2018-229909.

[10] Deutschland.de. *Germany as an industrialised country - the main facts*. 2023. URL: https://www.deutschland.de/en/topic/business/germanys-industry-the-most-important-facts-and-figures (visited on 01/15/2024).

[11] Mark Andrews. *How Digitization Is Changing the Way to Manufacture Cars*. 2022. URL: https://emag.directindustry.com/2022/07/25/how-digitization-is-changing-the-way-to-manufacture-cars/ (visited on 01/11/2024).

[12] European Parliament, Council of the European Union. *Regulation (EU) 2023/851 of the European Parliament and of the Council of 19 April 2023 amending Regulation (EU) 2019/631 as regards strengthening the CO2 emission performance standards for new passenger cars and new light commercial vehicles in line with the Union's increased climate ambition (Text with EEA relevance)*. Tech. rep. 2023/851. Available at https://eur-lex.europa.eu/eli/reg/2023/851/oj (visited 2024-01-30). 2023.

[13] Santokh Singh. *Critical Reasons for Crashes Investigated in the National Motor Vehicle Crash Causation Survey*. Brief Statistical Summary. 2018.

[14] Eleni Petridou and Maria Moustaki. "Human factors in the causation of road traffic crashes". In: *European Journal of Epidemiology* 16.9 (2000), pp. 819–826. DOI: 10.1023/A:1007649804201.

[15] Irene Overtoom, Gonçalo Correia, et al. "Assessing the impacts of shared autonomous vehicles on congestion and curb use: A traffic simulation study in The Hague, Netherlands". In: *International Journal of Transportation Science and Technology* 9.3 (2020), pp. 195–206. DOI: 10.1016/j.ijtst.2020.03.009.

[16] Jooyong Lee and Kara M Kockelman. "Energy implications of self-driving vehicles". In: *Proceedings of the 98th Annual Meeting of the Transportation Research Board, Washington, DC, USA*. 2019, pp. 13–17.

[17] Raphael van Kempen, Bastian Lampe, et al. "AUTOtech.agil: Architecture and Technologies for Orchestrating Automotive Agility". In: 32. Aachen Colloquium Sustainable Mobility. 2023. DOI: 10.18154/RWTH-2023-09783.

[18] Charlie Miller and Chris Valasek. "Remote exploitation of an unaltered passenger vehicle". In: *Black Hat USA* 2015.S 91 (2015). http://illmatics.com/Remote%20Car%20Hacking.pdf (visited on 2024-30-01).

[19] *Specification of Secure Onboard Communication*. Documentation Identification No: 969. AUTOSAR. 2020.

[20] Alexandru Kampmann, Bassam Alrifaee, et al. "A Dynamic Service-Oriented Software Architecture for Highly Automated Vehicles". In: *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*. 2019, pp. 2101–2108. DOI: 10.1109/ITSC.2019.8916841.

[21] McKinsey & Company. *Autonomous driving's future: Convenient and connected*. 2023. URL: https://www.mckinsey.com/industries/automotive-and-assembly/our-insights/autonomous-drivings-future-convenient-and-connected (visited on 01/11/2024).

[22] Steve Tengler. *Top 25 Auto Cybersecurity Hacks: Too Many Glass Houses To Be Throwing Stones*. 2020. URL: https://www.forbes.com/sites/stevetengler/2020/06/30/top-25-auto-cybersecurity-hacks-too-many-glass-houses-to-be-throwing-stones/?sh=5be0b54b7f65 (visited on 01/04/2024).

[23] KPMG. *Consumer Loss Barometer*. 2016. URL: https://assets.kpmg.com/content/dam/kpmg/cn/pdf/en/2016/08/consumer-loss-barometer-v1.pdf (visited on 01/04/2024).

[24] International Organization for Standardization. *ISO 11898-2:2016: Road vehicles Controller area network (CAN)*. Standard. Available at https://www.iso.org/standard/67244.html. 2016.

[25] International Organization for Standardization. *ISO 17458-1:2013 Road vehicles FlexRay communications system*. Standard. Available at https://www.iso.org/standard/59804.html. 2013.

[26] *SOME/IP Protocol Specification*. Documentation Identification No: 696. AUTOSAR. 2022.

[27] Object Management Group. *OMG Data Distribution Service*. 2015. URL: https://www.omg.org/spec/DDS/1.4/PDF (visited on 01/12/2024).

[28] Organization for the Advancement of Structured Information Standards. *Message Queuing Telemetry Transport*. 2014. URL: https://mqtt.org/ (visited on 03/02/2024).

[29] Object Management Group. *About the DDS Interoperability Wire Protocol Specification Version 2.5*. 2017. URL: https://www.omg.org/spec/DDSI-RTPS/ (visited on 02/03/2024).

[30] Object Management Group. *OMG DDS Security*. Version 1.1. 2018. URL: https://www.omg.org/spec/DDS-SECURITY/1.1/PDF (visited on 01/12/2024).

[31] Helena Handschuh. "Hardware-Anchored Security Based on SRAM PUFs, Part 1". In: *IEEE Security & Privacy* 10.3 (2012), pp. 80–83. DOI: 10.1109/MSP.2012.68.

[32] Nikolaos Athanasios Anagnostopoulos. "Practical Lightweight Security: Physical Unclonable Functions and the Internet of Things". PhD thesis. Technische Universität Darmstadt, 2022. DOI: 10.26083/tuprints-00021494.

[33] Zhiqiang Cai, Aohui Wang, et al. "0-days & mitigations: roadways to exploit and secure connected BMW cars". In: *Black Hat USA* 2019.39 (2019), p. 6.

[34] Sen Nie, Ling Liu, and Yuefeng Du. "Free-fall: Hacking tesla from wireless to can bus". In: *Briefing, Black Hat USA* 25.1 (2017). Available at https://www.blackhat.com/docs/us-17/thursday/us-17-Nie-Free-Fall-Hacking-Tesla-From-Wireless-To-CAN-Bus-wp.pdf (visited on 2024-30-01).

[35] ISO and SAE. ISO/SAE 21434:2021 road vehicles cybersecurity engineering. Available at https://www.iso.org/standard/70918.html. 2021.

[36] Christoph Schmittner and Georg Macher. "Automotive Cybersecurity Standards - Relation and Overview". In: *Computer Safety, Reliability, and Security*. 2019, pp. 153–165. DOI: 10.1007/978-3-030-26250-1_12.

[37] Marco Steger, Michael Karner, et al. "A security metric for structured security analysis of cyber-physical systems supporting SAE J3061". In: *2016 2nd International Workshop on Modelling, Analysis, and Control of Complex CPS (CPS Data)*. 2016. DOI: 10.1109/CPSData.2016.7496425.

[38] Christoph Schmittner, Zhendong Ma, et al. "Using SAE J3061 for Automotive Security Requirement Engineering". In: *Computer Safety, Reliability, and Security*. 2016, pp. 157–170. DOI: 10.1007/978-3-319-45480-1_13.

[39] *HEAling Vulnerabilities to ENhance Software Security and Safety (HEAVENS)*. 2018. URL: https://research.chalmers.se/en/project/5809 (visited on 11/15/2023).

[40] Shalabh Jain and Jorge Guajardo. "Physical Layer Group Key Agreement for Automotive Controller Area Networks". In: *Cryptographic Hardware and Embedded Systems – CHES 2016*. 2016, pp. 85–105.

[41] Stefan Nürnberger and Christian Rossow. "vatiCAN – Vetted, Authenticated CAN Bus". In: *Cryptographic Hardware and Embedded Systems – CHES 2016*. 2016, pp. 106–124. DOI: 10.1007/978-3-662-53140-2_6.

[42] Andreea-Ina Radu and Flavio D. Garcia. "LeiA: A Lightweight Authentication Protocol for CAN". In: *Computer Security – ESORICS 2016*, pp. 283–300. DOI: 10.1007/978-3-319-45741-3_15.

[43] Samir Fassak, Younes El Hajjaji El Idrissi, et al. "A secure protocol for session keys establishment between ECUs in the CAN bus". In: *2017 International Conference on Wireless Networks and Mobile Communications (WINCOM)*. 2017. DOI: 10.1109/WINCOM.2017.8238149.

[44] Paula Vasile, Bogdan Groza, and Stefan Murvay. "Performance Analysis of Broadcast Authentication Protocols on CAN-FD and FlexRay". In: *Proceedings of the WESS'15: Workshop on Embedded Systems Security*. 2015. DOI: 10.1145/2818362.2818369.

[45] Pal-Stefan Murvay and Bogdan Groza. "Practical Security Exploits of the FlexRay In-Vehicle Communication Protocol". In: *Risks and Security of Internet and Systems*. 2019, pp. 172–187. DOI: 10.1007/978-3-030-12143-3_15.

[46] Ahmed Refaat Mousa, Pakinam NourElDeen, et al. "Lightweight Authentication Protocol Deployment over FlexRay". In: *Proceedings of the 10th International Conference on Informatics and Systems (INFOS '16)*. 2016, pp. 233–239. DOI: 10.1145/2908446.2908485.

[47] Ahmed Hazem and HA Fahmy. "LCAP - A Lightweight CAN Authentication Protocol for Securing In-Vehicle Networks". In: *10th Embedded Security in Cars Conference (ESCAR)*. 2012.

[48] Marco Iorio, Massimo Reineri, et al. "Securing SOME/IP for In-Vehicle Service Protection". In: *IEEE Transactions on Vehicular Technology* 69.11 (2020), pp. 13450–13466. DOI: 10.1109/TVT.2020.3028880.

[49] Daniel Zelle, Timm Lauser, et al. "Analyzing and Securing SOME/IP Automotive Services with Formal and Practical Methods". In: *Proceedings of the 16th International Conference on Availability, Reliability and Security*. DOI: 10.1145/3465481.3465748.

[50] Simon Meier, Benedikt Schmidt, et al. "The TAMARIN Prover for the Symbolic Analysis of Security Protocols". In: *Computer Aided Verification*. 2013, pp. 696–701. DOI: 10.1007/978-3-642-39799-8_48.

[51] Victor Mayoral-Vilches, Ruffin White, et al. "SROS2: Usable Cyber Security Tools for ROS 2". In: *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2022, pp. 11253–11259. DOI: 10.1109/IROS47612.2022.9982129.

[52] Maxim Friesen, Gajasri Karthikeyan, et al. "A comparative evaluation of security mechanisms in DDS, TLS and DTLS". In: *Kommunikation und Bildverarbeitung in der Automation*. 2020, pp. 201–216. DOI: 10.1007/978-3-662-59895-5_15.

[53] Yakov Rekhter and Tony Li. *Datagram Transport Layer Seurity Version 1.2*. RFC 6347. 2012. DOI: 10.17487/RFC6347.

[54] Jeremy Erickson, Shibo Chen, et al. "CommPact: Evaluating the Feasibility of Autonomous Vehicle Contracts". In: *2018 IEEE Vehicular Networking Conference (VNC)*. 2018. DOI: 10.1109/VNC.2018.8628319.

[55] Sotiris Nikoletseas, Grigorios Prasinos, et al. "Attack propagation in Networks". In: *Theory of Computing Systems* 36.5 (2003), pp. 553–574. DOI: 10.1007/s00224-003-1087-5.

[56] Steven Noel, Lingyu Wang, et al. "Measuring Security Risk of Networks Using Attack Graphs". In: *International Journal of Next-Generation Computing* 1.1 (2010), pp. 135–147.

[57] Sebastian Roschke, Feng Cheng, and Christoph Meinel. "High-quality attack graph-based IDS correlation". In: *Logic Journal of the IGPL* 21.4 (2012), pp. 571–591. DOI: 10.1093/jigpal/jzs034.

[58] Martin Salfer and Claudia Eckert. "Attack graph-based assessment of exploitability risks in automotive on-board networks". In: *ARES 2018*. 2018.

[59] Michael Krisper, Jürgen Dobaj, et al. "RISKEE: A Risk-Tree Based Method for Assessing Risk in Cyber Security". In: *Systems, Software and Services Process Improvement - 26th European Conference, EuroSPI 2019, Proceedings*. 2019, pp. 45–56. DOI: 10.1007/978-3-030-28005-5_4.

[60] Dan Keilhoff, Dennis Niedballa, et al. "UNICARagil – New architectures for disruptive vehicle concepts". In: *19. Internationales Stuttgarter Symposium*. 2019, pp. 830–842. DOI: 10.1007/978-3-658-25939-6_65.

[61] Michael Buchholz, Fabian Gies, et al. "Automation of the UNICARagil vehicles". In: *29th Aachen Colloquium Sustainable Mobility*. Vol. 2. Universität Ulm. 2020, pp. 1531–1560. DOI: http://dx.doi.org/10.18725/OPARU-34024.

[62] International Society of Automation. *ISA/IEC 62443 Security for Industrial Automation and Control Systems*. Standard. 2017.

[63] United Nations Economic Commission for Europe (UNECE). *UN Regulation No. 155*. 2021. URL: https://unece.org/sites/default/files/2023-02/R155e%20%282%29.pdf (visited on 01/04/2024).

[64] U.S. Department of Transportation. "Revised Departmental Guidance 2016: Treatment of the Value of Preventing Fatalities and Injuries in Preparing Economic Analyses". In: (2021). Accessed on August 8, 2023.

[65] R.W. Saaty. "The analytic hierarchy process—what it is and how it is used". In: *Mathematical Modelling* 9.3 (1987), pp. 161–176. DOI: https://doi.org/10.1016/0270-0255(87)90473-8.

[66] Sebastian Karlstorfer. "A Security Risk Analysis for a Fully Automated Road Vehicle". Master's thesis. University of Passau, 2023.

[67] Nataliya Shevchenko, Timothy A. Chick, et al. *Threat Modeling: A Summary of Available Methods*. Tech. rep. Available at https://insights.sei.cmu.edu/documents/569/2018_019_001_524597.pdf (visited 2024-01-30). Carnegie Mellon University Software Engineering Institute, 2018.

[68] Jonathan Petit and Steven E. Shladover. "Potential Cyberattacks on Automated Vehicles". In: *IEEE Transactions on Intelligent Transportation Systems* 16.2 (2015), pp. 546–556. DOI: 10.1109/TITS.2014.2342271.

[69] Lotfi ben Othmane, Rohit Ranchal, et al. "Incorporating attacker capabilities in risk estimation and mitigation". In: *Computers & Security* 51 (2015), pp. 41–61. DOI: 10.1016/j.cose.2015.03.001.

[70] Zeinab El-Rewini, Karthikeyan Sadatsharan, et al. "Cybersecurity challenges in vehicular communications". In: *Vehicular Communications* 23 (2020), p. 100214. DOI: 10.1016/j.vehcom.2019.100214.

[71] Miro Enev, Alex Takakuwa, et al. "Automobile Driver Fingerprinting." In: *Proc. Priv. Enhancing Technol.* 2016.1 (2016), pp. 34–50. DOI: 10.1515/popets-2015-0029.

[72] Danny Dolev and Andrew Yao. "On the security of public key protocols". In: *IEEE Transactions on Information Theory* 29.2 (1983), pp. 198–208. DOI: 10.1109/TIT.1983.1056650.

[73] Anthony Van Herrewege, Dave Singelée, and Ingrid M. R. Verbauwhede. "CANAuth - A Simple, Backward Compatible Broadcast Authentication Protocol for CAN bus". In: *Conference: ECRYPT Workshop on Lightweight Cryptography*. 2011, pp. 229–235.

[74] BlackBerry. *Explaining Implicit Certificates*. URL: https://www.certicom.com/content/certicom/en/code-and-cipher/explaining-implicit-certificate.html (visited on 12/16/2023).

[75] Certicom Research. *Standards for Efficient Cryptography: SEC 4 - Elliptic Curve Qu-Vanstone Implicit Certificate Scheme (ECQV)*. Tech. rep. Version 1.0. Certicom Research, 2013.

[76] Kristin E Lauter and Katherine E Stange. "The elliptic curve discrete logarithm problem and equivalent hard problems for elliptic divisibility sequences". In: *International Workshop on Selected Areas in Cryptography*. Springer. 2008, pp. 309–327. DOI: 10.1007/978-3-642-04159-4_20.

[77] Nikolaos Athanasios Anagnostopoulos, Tolga Arul, et al. "Low-Temperature Data Remanence Attacks Against Intrinsic SRAM PUFs". In: *2018 21st Euromicro Conference on Digital System Design (DSD)*. 2018, pp. 581–585. DOI: 10.1109/DSD.2018.00102.

[78] Dennis K. Nilsson, Ulf E. Larson, et al. "A First Simulation of Attacks in the Automotive Network Communications Protocol FlexRay". In: *Proceedings of the International Workshop on Computational Intelligence in Security for Information Systems CISIS'08*. 2009, pp. 84–91. DOI: 10.1007/978-3-540-88181-0_11.

[79] Charles C. Y. Lam, Guang Gong, and Scott A. Vanstone. "Message Authentication Codes with Error Correcting Capabilities". In: *Proceedings of the 4th International Conference on Information and Communications Security*. 2002, 354–366. DOI: 10.1007/3-540-36159-6_30.

[80] Elena Dubrova, Mats Näslund, and Göran Selander. "CRC-Based Message Authentication for 5G Mobile Technology". In: *2015 IEEE Trustcom/BigDataSE/ISPA*. Vol. 1. 2015, pp. 1186–1191. DOI: 10.1109/Trustcom.2015.503.

[81] H. Krawczyk and P. Eronen. *HMAC-based Extract-and-Expand Key Derivation Function (HKDF)*. Request for Comments 5869. RFC Editor, 2010. DOI: 10.17487/RFC5869.

[82] Qiyan Wang and Sanjay Sawhney. "VeCure: A Practical Security Framework to Protect the CAN Bus of Vehicles". In: *2014 International Conference on the Internet of Things (IOT)*. 2014, pp. 13–18. DOI: 10.1109/IOT.2014.7030108.

[83] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 2001. ISBN: 0-8493-8523-7.

[84] Mihir Bellare. "New Proofs for NMAC and HMAC: Security Without Collision-Resistance". In: *Advances in Cryptology - CRYPTO 2006*, pp. 602–619. DOI: 10.1007/11818175_36.

[85] National Institute of Standards and Technology (NIST). *Recommendation for Key Management: Part 1 – General*. Tech. rep. NIST SP 800-57 Part 1 Rev. 5. National Institute of Standards and Technology, 2016. DOI: 10.6028/NIST.SP.800-57pt1r5.

[86] Mark Randolph and William Diehl. "Power Side-Channel Attack Analysis: A Review of 20 Years of Study for the Layman". In: *Cryptography* 4.2 (2020). DOI: 10.3390/cryptography4020015.

[87] Goncalo Martins, Arul Moondra, et al. "Computation and Communication Evaluation of an Authentication Mechanism for Time-Triggered Networked Control Systems". In: *Sensors* 16.8 (2016). DOI: 10.3390/s16081166.

[88] Hsin-Te Wu, Alan Yein, and Wen-Shyong Hsieh. "Message Authentication Mechanism and Privacy Protection in the Context of Vehicular Ad Hoc Networks". In: *Mathematical Problems in Engineering* 2015 (2015), pp. 1–11. DOI: 10.1155/2015/569526.

[89] Chen Yan, Wenyuan Xu, and Jianhao Liu. "Can you trust autonomous vehicles: Contactless attacks against sensors of self-driving vehicle". In: *Def Con* 24.8 (2016), p. 109.

[90] i11, RWTH Aachen. *embeddedRTPS*. https://github.com/embedded-software-laboratory/embeddedRTPS/commit/68915c8ccc744db171a202182dce2ef55b7728ca. 2022.

[91] Victor Bandur, Gehan Selim, et al. "Making the Case for Centralized Automotive E/E Architectures". In: *IEEE Transactions on Vehicular Technology* 70.2 (2021), pp. 1230–1245. DOI: 10.1109/TVT.2021.3054934.

[92] Bundesregierung Deutschland. *Entwurf eines Gesetzes zur Änderung des Straßenverkehrsgesetzes und des Pflichtversicherungsgesetzes – Gesetz zum autonomen Fahren*. 2021. URL: https://dserver.bundestag.de/btd/19/274/1927439.pdf (visited on 02/02/2023).

[93] Johannes Feiler, Simon Hoffmann, and Dr. Frank Diermeyer. "Concept of a Control Center for an Automated Vehicle Fleet". In: *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*. 2020. DOI: 10.1109/ITSC45102.2020.9294411.

[94] Leslie Lamport. "Password authentication with insecure communication". In: *Communications of the ACM* 24.11 (1981), pp. 770–772. DOI: 10.1145/358790.358797.

[95] B. Kaliski. *PKCS 5: Password-Based Cryptography Specification Version 2.0*. Tech. rep. 2898. 2000. DOI: 10.17487/RFC2898.

[96] C. Percival and S. Josefsson. *The scrypt Password-Based Key Derivation Function*. RFC. 2016. DOI: 10.17487/RFC7914.

[97] Christoph Böhm, Maximilian Hofer, and Wolfgang Pribyl. "A microcontroller SRAM-PUF". In: *2011 5th International Conference on Network and System Security*. 2011, pp. 269–273. DOI: 10.1109/ICNSS.2011.6060013.

[98] Andre Schaller, Wenjie Xiong, et al. "Intrinsic Rowhammer PUFs: Leveraging the Rowhammer effect for improved security". In: *IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. 2017. DOI: 10.1109/HST.2017.7951729.

[99] Jack Miskelly and Máire O'Neill. "Fast DRAM PUFs on Commodity Devices". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 39.11 (2020), pp. 3566–3576. DOI: 10.1109/TCAD.2020.3012218.

[100] eProsima. *Fast DDS*. https://github.com/eProsima/Fast-DDS/tree/627277859f6a5401c2bd236b4079e312e2580374. 2023.

[101] Adam Langley, W Chang, et al. *ChaCha20-Poly1305 cipher suites for transport layer security (TLS)*. RFC. 2016. DOI: 10.17487/RFC7905.

[102] Daniel J Bernstein. "The Poly1305-AES message-authentication code". In: *International Workshop on Fast Software Encryption*. Springer. 2005, pp. 32–49. DOI: 10.1007/11502760_3.

[103] A. Langley Y. Nir Dell EMC and Google Inc. *ChaCha20 and Poly1305 for IETF Protocols*. RFC. 2018. DOI: 10.17487/RFC8439.

[104] S. Turner A. Langley M. Hamburg. *Elliptic Curves for Security*. RFC. 2016. DOI: 10.17487/RFC7748.

[105] Craig Costello and Patrick Longa. *FourQ: four-dimensional decompositions on a Q-curve over the Mersenne prime*. Cryptology ePrint Archive, Paper 2015/565. https://eprint.iacr.org/2015/565. 2015.

[106] Abhranil Maiti and Patrick Schaumont. "Improved ring oscillator PUF: An FPGA-friendly secure primitive". In: *Journal of Cryptology* 24 (2011), pp. 375–397. DOI: 10.1007/s00145-010-9088-4.

[107] Jiliang Zhang, Gang Qu, et al. "A Survey on Silicon PUFs and Recent Advances in Ring Oscillator PUFs". In: *Journal of Computer Science and Technology* 29 (2014), pp. 664–678. DOI: 10.1007/s11390-014-1458-1.

[108] Florian Frank, Tolga Arul, et al. "Using Memristor Arrays as Physical Unclonable Functions". In: *Computer Security – ESORICS 2022*. 2022, pp. 250–271. DOI: 10.1007/978-3-031-17143-7_13.

[109] Johannes Feiler, Simon Hoffmann, and Dr. Frank Diermeyer. "Concept of a Control Center for an Automated Vehicle Fleet". In: *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*. 2020. DOI: 10.1109/ITSC45102.2020.9294411.

[110] George Coker, Joshua Guttman, et al. "Principles of remote attestation". In: *International Journal of Information Security* 10 (2011), pp. 63–81. DOI: 10.1007/s10207-011-0124-7.

[111] Carlton Shepherd, Konstantinos Markantonakis, and Georges-Axel Jaloyan. "LIRA-V: Lightweight Remote Attestation for Constrained RISC-V Devices". In: *2021 IEEE Security and Privacy Workshops (SPW)*. 2021, pp. 221–227. DOI: 10.1109/SPW53761.2021.00036.

[112] Ivan De Oliveira Nunes, Karim Eldefrawy, et al. "VRASED: A Verified Hardware/Software Co-Design for Remote Attestation". In: *28th USENIX Security Symposium (USENIX Security 19)*. 2019, pp. 1429–1446.

[113] Vittorio Todisco, Stefania Bartoletti, et al. "Performance Analysis of Sidelink 5G-V2X Mode 2 Through an Open-Source Simulator". In: *IEEE Access* 9 (2021), pp. 145648–145661. DOI: 10.1109/ACCESS.2021.3121151.

[114] *Intelligent Transport Systems (ITS); Access Layer; Part 1: Channel Models for the 5,9 GHz frequency band*. Technical Report. European Telecommunications Standards Institute, 2019.

[115] Andrea Baiocchi, Ion Turcanu, et al. "Age of Information in IEEE 802.11p". In: *2021 IFIP/IEEE International Symposium on Integrated Network Management (IM)*. 2021, pp. 1024–1031.

[116] Mahdi Dibaei, Xi Zheng, et al. *An Overview of Attacks and Defences on Intelligent Connected Vehicles*. 2019. DOI: 10.48550/arXiv.1907.07455.

[117] Hyun Min Song, Ha Rang Kim, and Huy Kang Kim. "Intrusion detection system based on the analysis of time intervals of CAN messages for in-vehicle network". In: *2016 International Conference on Information Networking (ICOIN)*. 2016, pp. 63–68. DOI: 10.1109/ICOIN.2016.7427089.

[118] ISO and IEC. *ISO/IEC 27005:2018 Information security*. Standard. Available at https://www.iso.org/standard/75281.html. 2018.

[119] Microsoft Corporation. *STRIDE: A Threat Modeling Framework*. 2009. URL: https://docs.microsoft.com/en-us/previous-versions/commerce-server/ee823878(v=cs.20) (visited on 01/15/2024).

[120] International Organization for Standardization. *ISO 15403-1:2022 Information security, cybersecurity and privacy protection*. Standard. Available at https://www.iso.org/standard/72891.html. 2022.

[121] ISO and IEC. *ISO/IEC 18045: Methodology for IT security evaluation*. Standard. Available at https://www.iso.org/standard/72889.htm. 2020.

[122] Martin Salfer and Claudia Eckert. "Attack Graph-Based Assessment of Exploitability Risks in Automotive On-Board Networks". In: *Proceedings of the 13th International Conference on Availability, Reliability and Security*. 2018. DOI: 10.1145/3230833.3230851.

[123] Patrick Helmholz, Edgar Ziesmann, and Susanne Robra-Bissantz. "Context-Awareness in the Car: Prediction, Evaluation and Usage of Route Trajectories". In: *Design Science at the Intersection of Physical and Virtual Design*. 2013, pp. 412–419. DOI: 10.1007/978-3-642-38827-9_30.

[124] Thomas H. Cormen, Charles E. Leiserson, et al. *Introduction to Algorithms*. 2nd. MIT Press, 2001, p. 599. ISBN: 978-0262032933.

A. Security Requirement Analysis

Criterion	Rating	Description
Safety	severe (4)	The passenger's death is likely.
	major (3)	Severe injuries like internal bleeding and brain damage are likely.
	moderate (2)	Light injuries like broken bones are possible.
	negligible (1)	No injuries or scratches are likely.
Financial	severe (4)	$]\$10M, \infty]$
	major (3)	$]\$10K, \$10M[$
	moderate (2)	$]\$0, \$10K]$
	negligible (1)	\$0
Operational	massive (4)	All traffic comes to an halt.
	medium (3)	A traffic jams occurs in a dedicated area.
	low (2)	A vehicle can operate only at reduced speed.
	none (1)	No restriction
Privacy	severe (4)	The leaked information reveals the passenger's identity.
	major (3)	The leaked data can be linked to the passenger at low cost.
	moderate (2)	The leaked sensitive data can only be linked to the passenger if available in large quantities.
	negligible (1)	Leaked data is not sensitive.

Table A.1: Impact criteria with their normalized and weighted scores

No	Asset	Impact Criteria				Impact	Likelihood Criteria						Likelihood	Risk	Zone
		PS	FL	OR	P		IC	WC	U	EI	B	TP			
1	sensor module	2	2	3	4	0,62	1	0	1	1	1	0	0.71	4	1
2	cerebrum	2	2	3	3	0,59	1	0	1	1	1	0	0.71	4	1
3	brainstem	2	2	3	2	0,72	1	0	1	1	1	0	0.71	4	1
4	dynamic module	4	3	4	1	0,85	1	0	1	1	1	0	0.71	5	2
5	lidar	1	2	2	3	0,42	0	0	0	0	1	1	0.11	1	3
6	radar	1	2	2	3	0,42	0	0	0	0	1	1	0.11	1	3
7	camera	1	2	3	4	0,51	0	0	0	0	1	1	0.11	2	3
8	hmi	2	2	2	4	0,56	1	1	1	1	1	0	0.96	4	4
9	localization	3	3	2	4	0,72	1	0	1	0	1	0	0.61	4	1
10	platform sensors	2	2	3	2	0,56	0	0	0	0	1	1	0.11	2	4
11	energy control system	3	3	3	1	0,69	1	0	1	1	1	0	0.61	3	
12	thermal control system	2	2	4	1	0,58	1	0	1	1	1	0	0.61	4	4
13	door control system	2	2	4	1	0,58	1	0	1	1	1	0	0.61	4	4
14	switch	3	3	3	1	0,69	1	0	1	1	1	1	0.75	4	A
15	router	3	3	3	4	0,78	1	0	1	1	1	1	0.75	5	A
16	control room	4	4	4	4	1	1	1	1	1	0	0	0.89	5	5
17	drone	1	2	1	3	0,37	1	1	1	0	0	0	0.79	3	5
18	cloud	1	1	2	3	0,37	1	1	1	0	0	0	0.79	3	5

Table A.2: Following identifying assets, a diverse panel of experts rated distinct impact and likelihood criteria, subsequently weighted through the Analytic Hierarchy Process (AHP). Impact and likelihood indicate the expected consequences and the probability of attacking a given asset. They eventually result in a risk value that assists in determining the zones and conduits used in the detailed risk analysis.

Top Level Threat	ID	Intermediate Threat	ID	Zones	Risk
Manipulation of driving behavior	t_0	Attacker manipulates control variables on FlexRay bus	$t_{0,0}$	C_2	very high
		Firmware corruption of dynamic modules	$t_{0,1}$	Z_2	high
		Attacker manipulates control variables in ASOA	$t_{0,2}$	C_1	very high
		Impersonation of the control room	$t_{0,3}$	C_4	moderate
		Forgery of control room commands	$t_{0,4}$	C_4	high
Random opening/closing of the door	t_1	Manipulation of door control firmware	$t_{1,0}$	Z_4	high
		Manipulation of brainstem firmware	$t_{1,1}$	Z_1	very high
		Injection of false opening/closing command to Ethernet bus	$t_{1,2}$	C_1	very high
Camera data forgery	t_2	Manipulation of camera data on the Ethernet bus	$t_{2,0}$	C_1	very high
		Impersonation of camera	$t_{2,1}$	C_1	very high
		Manipulation of camera firmware	$t_{2,2}$	Z_4	moderate
		DoS attack on Ethernet bus	$t_{2,3}$	C_1	very high
Lidar data forgery	t_3	Manipulation of lidar commands on Ethernet bus	$t_{3,0}$	C_1	very high
		Impersonation of lidar	$t_{3,1}$	C_1	very high
		Manipulation of lidar firmware	$t_{3,2}$	Z_4	moderate
		DoS attack on Ethernet bus	$t_{3,3}$	C_1	very high
Radar data forgery	t_4	Manipulation of ASOA radar commands on CAN bus	$t_{4,0}$	C_3	high
		Impersonation of camera	$t_{4,1}$	C_3	very high
		Manipulation of camera firmware	$t_{4,2}$	Z_3	moderate
		DoS attack on Ethernet bus	$t_{4,3}$	C_1	very high
Eavesdropping of position data	t_5	Spoofing of the localization module	$t_{5,0}$	C_1	high
		Firmware corruption of the localization module	$t_{5,1}$	Z_1	high
Deliberate safe halt	t_6	Firmware corruption of brainstem	$t_{6,0}$	Z_1	very high
		Fake environment perception	see threats t_2 - t_4		
		DoS attack on Ethernet bus	$t_{6,2}$	C_1	very high
Battery corruption	t_7	Firmware manipulation of energy module	$t_{7,0}$	Z_4	very high
		Manipulation of ASOA commands on Ethernet bus	$t_{7,1}$	C_1	very high
Forgery of the external environment model	t_8	Identity theft of vehicle	$t_{8,0}$	C_4, Z_1	very high
		Identity theft of cloud	$t_{8,1}$	C_4, Z_6	very high
		Manipulation of data in the cloud	$t_{8,2}$	Z_6	moderate
		Manipulation of environment data from cloud to vehicle	$t_{8,3}$	C_4	very high

Forging traffic from drone	t_9	Identity theft of drone	$t_{9,0}$	C_4, Z_6	moderate
		Manipulation of drone commands	$t_{9,1}$	C_4	very high
Remote vehicle hijacking	t_{10}	Impersonation of the control room	$t_{10,0}$	C_4, Z_4	very high
		Manipulation of remote control commands	$t_{10,1}$	C_4	very high
		Hijacking of the control room	$t_{10,2}$	Z_5	moderate
User deception	t_{11}	Firmware corruption of HMI	$t_{11,0}$	Z_4	high
		Manipulation of ASOA control commands on Ethernet bus	$t_{11,1}$	C_1	very high
Position manipulation	t_{12}	GPS jamming	$t_{12,0}$	C_4	high
		Firmware corruption of the localization module	$t_{12,1}$	Z_1	high
		Manipulation of position on Ethernet bus	$t_{12,2}$	C_1	very high
Forgery of environmental model	t_{13}	Firmware corruption of the cerebrum	$t_{13,1}$	Z_1	very high
		Manipulation of environment data from cloud to vehicle	$t_{13,2}$	C_4	very high
Log Stealing	t_{14}	Illegal access to sensitive logs on the main HPC	$t_{14,1}$	Z_1	very high
		Illegal access to sensitive logs low-level control units	$t_{14,2}$	Z_4	very high
Privilege Escalation	t_{15}	Malicious application on HPCs interferes with other applications	$t_{15,1}$	Z_1	very high
		Resource Exhaustion on HPCs	$t_{15,2}$	Z_1	very high

Table A.3: Complete overview of all threats used in the security requirement analysis

Intermediate Threat	**Impact**				**Likelihood**				**Risk**
					Capabilities			**Vuln.**	
	PS	FL	OR	P	Ex	K	R		
$t_{0,0}$	3	3	3	1	2	2	2		high
$t_{0,1}$	4	3	3	1	2	1	2		high
$t_{0,2}$	4	3	3	2	3	2	3		very high
$t_{0,3}$	4	4	4	3	2	2	2		high
$t_{0,4}$	4	3	3	2	3	2	3		very high
$t_{1,0}$	3	2	2	1	2	1	2		high
$t_{1,1}$	4	3	3	4	2	1	2		high
$t_{1,2}$	3	2	2	1	3	2	3		very high
$t_{2,0}$	4	3	3	1	3	3	3		very high
$t_{2,1}$	4	3	3	1	2	2	2		high
$t_{2,2}$	4	3	3	3	2	1	2		high
$t_{2,3}$	4	3	3	1	3	3	3		very high
$t_{3,0}$	4	3	3	1	3	2	3		very high
$t_{3,1}$	4	3	3	1	2	2	2		high
$t_{3,2}$	4	3	3	1	2	2	2		high
$t_{3,3}$	4	3	3	1	3	3	3		high
$t_{4,0}$	4	3	3	1	2	1	2		high
$t_{4,1}$	4	3	3	1	2	2	2		high
$t_{4,2}$	4	3	3	3	2	1	2		high
$t_{4,3}$	4	3	3	1	3	3	3		very high
$t_{5,0}$	3	2	2	1	3	2	3	medium	very high
$t_{5,1}$	3	3	3	4	2	1	2		high
$t_{6,0}$	4	3	3	4	2	1	2		high
$t_{6,1}$	4	3	3	1	3	3	3		very high
$t_{7,0}$	4	3	3	1	2	1	2		high
$t_{7,1}$	4	3	3	2	3	2	3		very high
$t_{8,0}$	2	2	1	4	2	3	3		very high
$t_{8,1}$	3	3	3	4	2	2	3		very high
$t_{8,2}$	3	4	4	2	1	1	2		high
$t_{8,3}$	3	3	3	1	2	2	2		high
$t_{9,0}$	0	0	0	0	0	0	0		very high
$t_{9,1}$	0	0	0	0	0	0	0		very high
$t_{10,0}$	4	4	4	3	2	2	2		high
$t_{10,1}$	4	3	3	2	2	2	2		high
$t_{10,2}$	4	4	4	4	2	1	1		high
$t_{11,0}$	3	2	2	4	2	1	2		high
$t_{11,1}$	4	3	3	2	3	2	3		very high
$t_{12,0}$	2	1	3	1	3	3	3		major
$t_{12,1}$	0	0	0	0	0	0	0		very high
$t_{12,2}$	4	3	3	1	3	3	3		very high
$t_{13,1}$	4	3	3	4	2	1	2		high
$t_{13,2}$	3	3	3	1	2	2	2		high
$t_{14,1}$	0	0	0	4	2	2	2		moderate
$t_{14,2}$	0	0	0	4	3	3	3		high
$t_{15,1}$	4	3	3	2	2	1	2		very high
$t_{15,2}$	3	3	3	1	2	2	2		high

Table A.4: The impact and likelihood of each intermediate threat is assessed individually.

B. Securing Service-Oriented Architectures

```
theory token_distribution
begin

builtins: diffie-hellman, symmetric-encryption, signing
functions: mac/2
equations: sdec(senc(m, k), k) = m

// Initialization and deployment rules

// The OnlyOnce() action ensures that the keys are generated only once.
    Otherwise, the attacker can re-create keys such that the lemmas cannot be
    proven any more.
rule Deploy_ID_Keys:
 [Fr(~id_key)]
 --[OnlyOnce(),
    Has_ID_Key_ECU(~id_key), Has_ID_Key_ASP(~id_key)
   ]->
 [!ID_Key_ECU(~id_key), !ID_Key_ASP(~id_key)]

rule Create_SK:
 [Fr(~sk)]
 -->
 [!SK($A, ~sk)]

// Protocol steps

rule ecu_send_token_request:
 let
  ecupubkey = 'g'^sk
  req = <$ECU, ~ecunonce, ecupubkey>
 in
  [Fr(~ecunonce), !ID_Key_ECU(id_key), !SK($ECU, sk)]
  --[Send_1($ECU, req), Secret(id_key), Secret(sk)
    ]->
  [Out(<req, mac(req, id_key)>)]

rule asp_receive_request_send_tokens:
 let
  asppubkey = 'g'^sk
  rcdash = ecupubkey^sk
  enc_tokens = senc(~tokens, rcdash)
  enc_nonce = senc(~aspnonce, rcdash)
  req = <$ECU, ecunonce, ecupubkey>
  reply = <$ASP, asppubkey, enc_nonce, enc_tokens>
 in
```

```
44  [Fr(~aspnonce), Fr(~tokens), !SK($ASP, sk), !ID_Key_ASP(id_key), In(<req, mac(
        req, id_key)>)]
45  --[Recv_1($ASP, req),  Send_2($ASP, reply),  Authentic($ECU, req), Secret_Nonce
        (~aspnonce),  Secret(~tokens), Freshness(ecunonce), Secret(sk), Secret(
        id_key), Tokens_Created($ECU, ~tokens),   ASP_Update_Key($ASP, rcdash)
46    ]->
47   [Step_2($ASP, ~aspnonce, ~tokens), Out(reply)]
48
49  rule ecu_receive_tokens_send_nonce:
50   let
51    rcdash = asppubkey^sk
52    dec_nonce = sdec(enc_nonce, rcdash)
53    dec_tokens = sdec(enc_tokens, rcdash)
54    reply = <$ASP, asppubkey, enc_nonce, enc_tokens>
55   in
56   [Step_2($ASP, aspnonce, tokens), !ID_Key_ECU(id_key), !SK($ECU, sk), In(reply)]
57   --[Recv_2($ECU, reply), Send_3($ECU, dec_nonce), ECU_Update_Key($ECU, rcdash),
        Secret(id_key), Secret(sk), Nonce_Revealed(aspnonce), Eq(aspnonce, dec_nonce
        ), Eq(tokens, dec_tokens), Tokens_Received($ECU, tokens)
58    ]->
59   [Out(<$ECU, dec_nonce>)]
60
61  rule asp_receive_nonce:
62   [In(<$ECU, aspnonce>)]
63   --[Recv_3($ASP, aspnonce),
64      Fin($ASP, aspnonce)
65    ]->
66   []
67
68  // Restrictions
69
70  restriction UniqueNonce:
71   "not(Ex n1 n2 #i #j. Freshness(n1)@#i & Freshness(n2)@#j & n1 = n2 & not(#i = #
        j))"
72
73  restriction OnlyOnce:
74   "All #i #j. OnlyOnce()@#i & OnlyOnce()@#j ==> #i = #j"
75
76  restriction Equality:
77   "All x y #i. Eq(x,y) @i ==> x = y"
78
79  // Security Lemmas
80
81  // This lemma proves the authentic transmission of ECU's public key to the ASP.
82  // Without this guarantee, an attacker could perform a MitM attack and forge the
        computation of the identity key.
83  lemma Authentic_ECU_PK:
84   "All b m #i. Authentic(b,m) @i
85   ==> (Ex #j. Send_1(b,m) @j & j<i)"
86
87  lemma Fresh_ECU_PK:
88   "All n1 n2 #i #j. (Freshness(n1)@i & Freshness(n2)@j & not(#i = #j) ==> not(n1
        = n2))"
89
90  // This lemma proves that the attacker never learns confidential data that is
       put on the wire. This includes the security tokens and the aspnonce until it
       is published by the ECU.
91  lemma Secret_Transmission:
```

```
92  "(All x #i.
93  Secret(x) @i ==> not (Ex #j. K(x)@j))  &
94  (All x #i.
95  (Secret_Nonce(x)@i & not(Nonce_Revealed(x)@i)) ==> (not(Ex #j. K(x)@j & (j < i)
        ) & not(K(x)@i)))"
96
97  // This lemma proves that both the ECU and the ASP eventually share the same
        secret tokens. For an attacker, it is impossible to forge them during
        transmission.
98  lemma Same_Tokens:
99  "All ecu  x #i #j .
100 Tokens_Created(ecu, x)@i & Tokens_Received(ecu, x)@j  ==> not(Ex #k. K(x)@k & j
        < k) & i < j "
101
102 // This lemma proves that the attacker never learns the identity key and
        therefore is not able to impersonate any legitimate ECU
103 lemma ID_Key_Never_Leaked:
104 "(All ECU rcdash #i.
105 ECU_Update_Key(ECU, rcdash)@i ==> not (Ex #j. K(rcdash)@j)) &
106 (All ASP rcdash #i.
107  ASP_Update_Key(ASP, rcdash)@i ==> not (Ex #j. K(rcdash)@j)) &
108 (All id_key #i.
109  Has_ID_Key_ASP(id_key)@i ==> not (Ex #j. K(id_key)@j)) &
110 (All id_key #i.
111  Has_ID_Key_ECU(id_key)@i ==> not (Ex #j. K(id_key)@j))"
112
113 // Operational Lemmas
114
115 // This lemma proves that the protocol is executable and logically well defined.
116 lemma Executable_and_Terminable:
117 exists-trace
118 "Ex ECU ASP request reply ack #req_sent #req_recv #reply_sent #reply_recv #
        ack_sent #ack_recv.
119 (Send_1(ECU,request)@ #req_sent & Recv_1(ASP,request)@ #req_recv &
120 Send_2(ASP,reply)@ #reply_sent & Recv_2(ECU,reply)@ #reply_recv &
121 Send_3(ECU,ack)@ #ack_sent & Recv_3(ASP,ack) @ #ack_recv &
122 req_sent < req_recv &
123 req_recv < reply_sent &
124 reply_sent < reply_recv &
125 reply_recv < ack_sent &
126 ack_sent < ack_recv)
127 "
128
129 // This lemma proves that the ECU only updates its identity key after the ASP
        has already computed the new key.
130 lemma ASP_Update_After_ECU_Update:
131 "All ecu rcdash #i.
132  ECU_Update_Key(ecu, rcdash) @ i
133  ==> (Ex asp #j. ASP_Update_Key(asp, rcdash) @ j & j < i)"
134
135 end
```

Listing B.1: This is the complete Tamarin proof of the proposed token distribution protocol depicted in Figure 6.6.