CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Device Attestation for In-Vehicle Network

Master's thesis in Computer science and engineering

Erik Andréasson, Ivan Lyesnukhin

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2022

Master's Thesis 2022

Device Attestation for In-Vehicle Network

Erik Andréasson, Ivan Lyesnukhin

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2022

ERIK ANDRÉASSON, University of Gothenburg
gusanderdk@student.gu.se

IVAN LYESNUKHIN, Chalmers University of Technology
livan@student.chalmers.se

Supervisor: Roger Johansson, CSE, roger@chalmers.se
Industry advisor: Nasser Nowdehi, Volvo Cars, nasser.nowdehi@volvocars.com
Examiner: Tomas Olovsson, CSE, tomas.olovsson@chalmers.se

Master's Thesis 2022
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Gothenburg, Sweden 2022

Abstract

Embedded devices are crucial components of all modern vehicles. Today, every in-vehicle architecture consists of multiple small devices, each of which is responsible for a limited number of tasks, and together the devices act as a whole, giving the vehicle a big range of functionalities. In recent years, embedded devices have shown to be highly vulnerable to cyberattacks which put the driver at risk, as the vehicle could be controlled by an unauthorized party. Therefore, securing the vehicle's embedded devices from unauthorized use and modification is very important. One of the ways to protect the embedded devices is to perform a so-called *device attestation*, which is a technique used to certify that a specific firmware is unchanged and valid. This technique has been proven to work outside of the automotive industry, but the main challenge is to get it to work with the strict requirements that exist on vehicular hardware. There are many ways of performing device attestation and these approaches can have different impact on cost, efficiency, level of security, memory capacity used, amount of hardware needed, and possibly other factors. This paper classifies a number of device attestation techniques into different categories and discusses what effect these techniques have on the cost of installation, security level, and efficiency. It also describes how these methods can secure embedded devices inside the vehicle and how applicable the methods are for the modern in-vehicle architecture. Further, the report proposes device attestation methods that satisfy three main criteria: the solution should be applicable for modern vehicles, come with a low cost, and meet the automotive-specific security and safety requirements.

Keywords: attestation, device attestation, cybersecurity, vehicular security, automotive security.

Acknowledgements

We would like to express our gratitude to all the people and organizations that have been involved and engaged in this thesis, and who helped and supported us during this project. The Chalmers University of Technology and Volvo Cars Corporation both gave us the knowledge and possibility to perform this research project. Our industrial supervisor, Nasser Nowdehi, at Volvo Cars has shown a big enthusiasm about the thesis and has been highly supportive throughout the whole project. We also got great support from our academic supervisor Roger Johansson and examiner Tomas Olovsson at Chalmers and Gothenburg University.

Erik Andréasson, Gothenburg, Sweden, July 2022
Ivan Lyesnukhin, Gothenburg, Sweden, July 2022

Contents

List of Figures **xi**

1 Introduction **1**
1.1 Background . . . 2
1.2 Related work . . . 3

2 Theory **5**
2.1 In-vehicle network . . . 5
2.2 Security fundamentals . . . 8
2.2.1 CIA - confidentiality, integrity, availability . . . 9
2.2.2 Encryption . . . 9
2.2.3 Checksum and Hash . . . 9
2.2.4 Attack types . . . 10
2.3 Device attestation . . . 12
2.3.1 Basics of device attestation . . . 12
2.3.2 Attestation types . . . 13
2.3.3 Physical unclonable function . . . 14
2.3.4 Key predistribution system . . . 15
2.3.5 Protected bootloader . . . 15
2.4 Requirements . . . 15
2.4.1 Attestation requirements . . . 16
2.4.2 Vehicular requirements on security and attestation . . . 16
2.5 Adversary model . . . 17

3 Method **19**
3.1 Literature review . . . 19
3.2 Proposed attestation methods . . . 20
3.3 Limitations . . . 20

4 Literature review results **21**
4.1 Master ECU approach . . . 21
4.1.1 Internal master ECU approach . . . 21
4.1.2 External master ECU approach . . . 23
4.2 Decentralized attestation approach . . . 26
4.3 External attestation approach . . . 32
4.4 Hardware attestation approach . . . 33
4.5 Software attestation approach . . . 33

4.5.1 Various software solutions . . . 33
4.5.2 Elimination of physical attacks . . . 37
4.6 Hybrid attestation approach . . . 38
4.7 Dynamic attestation approach . . . 40

5 Proposed attestation methods 43
5.1 When to perform attestation . . . 43
5.2 Attestation protocols . . . 44
5.2.1 Protocol 1 . . . 44
5.2.2 Protocol 2 . . . 45
5.3 Feasibility . . . 49

6 Discussion 51
6.1 Attestation methods and their applicability . . . 51
6.1.1 Master ECU approach . . . 51
6.1.2 Decentralized attestation approach . . . 52
6.1.3 External attestation approach . . . 54
6.1.4 Hardware attestation approach . . . 55
6.1.5 Software attestation approach . . . 55
6.1.6 Hybrid attestation . . . 57
6.1.7 Dynamic attestation approach . . . 57
6.2 Proposed attestation methods . . . 57
6.3 What to do if attestation fails? . . . 59
6.4 Trends in device attestation . . . 60
6.5 Alternative solutions for the future . . . 60

7 Conclusion & Future work 63

List of Figures

2.1 Example of IVN and ECU functionality 6
2.2 CAN bus and ECUs inside the vehicle 6
2.3 IVN network in modern vehicles. Each domain is a powerful ECU which is responsible for a number of simple ECUs with the related tasks 7
2.4 Future IVN architecture with a Central Unit. 8
2.5 Replay Attack. Adversary Eve captures Alice's hash and resends it later to Bob. 11
2.6 Remote Attestation Example 13
2.7 PUF setup phase. RNG is random generator, $C_{1...n}$ are challenges produced by random genrator, R is responce made by PUF. 14

4.1 ALE protocol - Master Attestation Protocol 23
4.2 Internal vehicle architecture for the protocol using master and remote Center 24
4.3 Master ECU approach using a remote verification Center. 26
4.4 Decentralized Attestation Approach 27
4.5 Salad protocol. 1. Initiation, 2. Generation of Attestation Reports, 3. Distributed Report Aggregation, 4. Attestation Finalization. . . . 29
4.6 Decentralized Attestation with spanning tree and Schnorr 31
4.7 Example of attack on SWATT [54] 34
4.8 Challenge-response in normal (left) and proxy attack (right) scenario. V is verifier, P is prover, X is proxy, n is nonce (challenge), csum is checksum 36
4.9 Challenge-response with VIPER in normal (left) and proxy attack (right) scenario. V is verifier, P is prover, X is proxy, n is nonce (challenge), csum is checksum 37
4.10 SMART protocol. V is verifier, P is prover. 42

5.1 Protocol 1 architecture. ECUs with SRAM are attested with AAoT and others are attested with VIPER 45
5.2 Protocol 2 - Phase 1 47
5.3 Protocol 2 - Second phase. ECU with red color is a critical ECU. . . 48
5.4 Protocol 2 - Phase 3 49

1
Introduction

Embedded systems have shown rapid advancement over the past few years and nowadays they are used in various spheres of life - whether it is inside an IoT device or an aircraft. An embedded system differs from an "ordinary" computer in the way that it often only has one single responsibility, at the same time as it strives for low cost and low power consumption [33]. Modern vehicles are examples where such systems are used.

Nowadays, cars consist of more than 100 Electronic Control Units (ECU) which build up an In-Vehicle Network (IVN). Today the ECUs control safety and security-critical parts such as engine, transmission, airbag and anti-lock brake system [31]. These ECUs have different software and hardware configurations and communicate with each other over the IVN using mainly the Controller Area Network, also known as the CAN bus, but the communication can also be performed via Ethernet, FlexRay, and Local Interconnect Network (LIN).

The ECUs bring a lot of enhancements when it comes to road safety and driving efficiency, but in recent years they have proved to be vulnerable to a variety of cyberattacks [60]. These vulnerabilities include that an adversary can remotely manipulate the code of the ECUs, thus taking control over critical parts of the vehicle, which eventually can have dangerous consequences. In order to avoid such a scenario, a method for verifying the software integrity of the ECUs is necessary. The method of verifying the firmware on the ECU or any other device is called *device attestation* or *remote device attestation*. In this thesis, the word *attestation* is mostly used and it refers to the concept of (remote) device attestation and the word *device* usually refers to the ECU. The attestation method needs to be able to be applied efficiently on all ECUs to detect whether there has been any modification to the code to make the vehicle and its passengers more secure and safe.

Researchers have tried to come up with secure and efficient ways of performing attestation on vehicles to verify their software's integrity. By vehicles, this paper mainly refers to cars if nothing else is mentioned, but the results of this research can partly be applied in other types of vehicle such as trucks or trains. A common approach of attesting the device's software is to apply already working attestation methods from other areas onto vehicular systems. Because of the specific and restricted functionality of vehicular embedded systems, the attestation methods should follow specific requirements in order to be applied in the vehicles. This results in many methods failing to meet these vehicle-specific requirements, thus also failing

to provide a sufficient attestation method. Two specific constraints that often are overseen or broken are the low cost and efficiency aspects. Many attestation methods tend to use expensive hardware, for example for securing cryptographic secrets or other information that is used in the authentication process of the attestation.

The questions to be answered in this project:

1. What is the current state of the art in remote attestation for embedded systems?
2. What are the automotive-specific challenges (e.g., with respect to scalability, performance, etc.) of adapting the existing attestation solutions to IVNs?
3. What attestation methods can be proposed which would be applicable to modern IVNs?

The main goal of this thesis is to give answers to the three above-mentioned questions. Also, the paper has a goal to categorize and describe a number of existing device attestation techniques. Further, it discuss techniques' applicability to embedded systems and how they can contribute to a secure in-vehicle network. In addition to this, the report aims to propose a device attestation method that satisfies three main criteria: the solution should be able to be applied in modern vehicles, come with a low cost and meet the automotive-specific security requirements.

The report is outlined as follows. Section 1.1 introduces the reader to some background to the project and Section 1.2 presents the existing work available at the moment. Section 2 familiarizes the reader with all theory needed to understand the rest of the report and Section 3 describes the approach taken in this project. Later on, Section 4 presents performed literature study by describing a number of methods of doing device attestation. Next, Section 5 describes proposed methods followed by Section 6 which will discuss the things found during the research. Finally, Section 7 talks about the conclusions of this project and discuss the possible future works that could be carried out on the basis of this project.

1.1 Background

Through the years there have been an increasing amount of cyberattacks aimed against modern vehicles. Car manufacturers such as Toyota, Jeep, BMW and Tesla have all fallen victim to remote attacks against their vehicles. In 2015, two researchers, Miller and Valasek proved there existed multiple ways of remotely attacking and breaching the security of a Jeep Cherokee [45]. Some of the attacks they successfully accomplished were, taking control over the infotainment system which implies that they could track the vehicle through its GPS, control the heating and air conditioning, control the media volume and change what is showing on the displays. Further, they managed to take control of the brakes and steering which could lead to devastating situations.

Similarly in 2018, the Tencent Keen Security Lab found equally severe vulnerabilities in several BMW models [40]. They managed to take control of the vehicle

both via physical and remote access. Not more than a year earlier, Jason Hughes managed to take control over an entire fleet of Tesla vehicles [24]. The hack let him take control over any Tesla and tell it to autonomously drive anywhere. Fortunately, Jason was a good guy and didn't want to cause any harm but imagine the consequence this could have if it falls into the wrong hands.

The main reason for this increase in attacks against vehicles is the increase in vehicle communication methods [61]. As it was mentioned above, modern vehicles today consist of a number of ECUs which are capable to communicate with the outside world through the Vehicular ad hoc network (VANET). This technique makes vehicle-to-vehicle (V2V), vehicle-to-infrastructure (V2I), and vehicle-to-IoT (V2IoT) communication possible. In this scenario, the possibility of creating a smart and efficient road network increases, but it also enlarges the attack surface. Opening up the vehicles for attacks can have potentially unwanted consequences, such as disclosure of information, loss of vehicle control, loss of property, and performance degradation.

It is not always that vehicles are hacked by an unauthorized adversary, sometimes vehicle owners intentionally "hack" their vehicle to unlock paid services for free or at a significantly lower cost. Today it is common for car manufacturers to make fewer physical changes between models, but instead, they lock features through software such as performance boosts and self-driving. The manufacturers then offer these features at an extra cost. In 2020, a third party offered Tesla owners to unlock a performance boost of 50 horsepowers at a low cost which Tesla offered for 2000 US$ [41]. Software modifications like this by an unauthorized party could easily result in unwanted behaviors and further put both the vehicle and driver at risk and of course the car manufacturer. Even if the moral behind locking features in a vehicle that has been bought by a customer can be discussed, tampering with the original software always comes with the risk of damaging the vehicle itself and others in its surrounding.

1.2 Related work

A number of research studies have been done in the area of device attestation. One significant work has been done by Florian Kohnhäuser in his PhD thesis [33]. This work discusses a number of attestation topics such as attestation of commodity low-end embedded systems, application attestation with embedded systems, attestation of highly dynamic networks, attestation of software and physical attacks and attestation of autonomous embedded systems.

Another survey [39] describes different remote attestation schemes from the perspective of their taxonomy in order to show the various characteristics of each remote attestation approach, its weaknesses, and potential improvements. The authors proposed a universal adversary model for IoT devices to support attestation taxonomy by describing various attacks against it. A similar paper [11] further elaborates on the possible attacks in combination with the limitations of a variety of attestation protocols.

Steiner et al. [58] did similar research, but with a focus on wireless sensor networks. The authors discussed a number of attestation approaches in wireless sensor networks as well as their assumptions, motivation, challenges and attacks. Additionally, the article describes a complex taxonomy of the attestation and the connection between it and each attestation approach.

Paper [8] describes collective attestation principles for large-scale and dynamic networks. The authors discuss the efficiency, scalability, as well as security of the attestation methods for a large network of provers who attest themself and cooperate to create a joint result.

Ioannis Sfyrakis and Thomas Gross in [57] did a thorough review of the state-of-the-art hardware-based attestation methods. The paper highlights different hardware-based methods for remote attestation which make use of TPMs to store cryptographic keys. A TPM is a specific hardware that is suitable for securely storing secret information but it is also a relatively expensive piece of hardware that is out of the picture in low-end devices where low cost is a high priority. Except for hardware solutions the survey touches upon hybrid solutions which could be used for low-end devices. The authors also presented a complex taxonomy of the hardware-based attestation methods. Since the work is limited to hardware and hybrid approaches it misses out on some relevant methods such as pure software-based ones. Further, the survey is not aimed at low-end embedded systems which are prominent in IVNs. Thus, vehicle-specific requirements are not taken into account in this survey.

Another research [6] discusses remote attestation on an IoT scale and specifically focuses on the adversary and attack typologies in combination with mitigation strategies. The paper thoroughly describes the hardware and hybrid attestation methods and their attacker models, but it lacks the vehicle-specific viewpoint where there are strict requirements on the hardware. Further, the article is mainly focusing on attestation methods applicable for swarms of devices which is good, but not the only type of attestation methods that are applicable to the vehicle industry. The authors also mention that most swarm attestation methods rely on specialized hardware, for example, to store cryptographic keys which is unlikely to exist in low-end vehicle ECUs.

As it can be seen there are a number of publications and research performed in the area of attestation. Most of the found surveys focus on specific areas of attestation, but no public survey with the focus on attestation of the in-vehicular systems has been done so far. This paper has the aim to collect and describe different attestation techniques in relation to the automobile industry, the vehicle's security, and the requirements.

2
Theory

This section introduces the theory that is needed to understand the context of this thesis. Section 2.1 gives basic knowledge about the in-vehicle network and how the communication between devices in the vehicle is structured. Later on, in Section 2.2, the reader is familiarized with security fundamentals and Section 2.3 introduces the basic concepts of device attestation. Section 2.4 describes security requirements on both the attestation methods and the vehicles. Finally, Section 2.5 outlines the adversary model by describing what assumptions and capabilities of the adversary were assumed in this project.

2.1 In-vehicle network

About 40 years ago vehicles were almost purely mechanical, but since then an increasing amount of computational power has been added year after year. As a result, nowadays vehicles consist of an entire In-vehicle Network and multiple embedded computers which are called Electronic Control Units. These ECUs can do anything from coordinate critical tasks, monitor sensors, monitor the people inside the vehicle, to assist the driver in difficult situations [36]. Modern vehicles are equipped with over 100 ECUs and this number is expected to rise in the upcoming years[48]. An ECU is not an ordinary computer with a high-performance CPU, generous storage capacity and a powerful GPU. It is actually the opposite: an ECU is designed to have a limited number of tasks, be as energy and time-efficient as possible and be produced at a low cost. These requirements are not things that go hand in hand with high security.

In Figure 2.1, some of the responsibilities of ECUs are displayed. As can be seen, the responsibilities can be non-crucial e.g. infotainment system, air conditioning and windows, or highly-critical e.g. airbag, engine, transmission and antilock braking system.

The in-vehicle network (IVN) is what ties all the ECUs together and works as a communication medium between them. The IVN is responsible for all the data transfer inside the vehicle and consists of the Controller Area Network (CAN), Local Interconnect Network, FlexRay and Ethernet[31]. Different communication protocols have different responsibilities and transfer different kinds of data. The CAN protocol consists of a bus to which ECUs are connected. Figure 2.2 gives a simple visualization of what the CAN bus and the ECUs connected to it look like. For

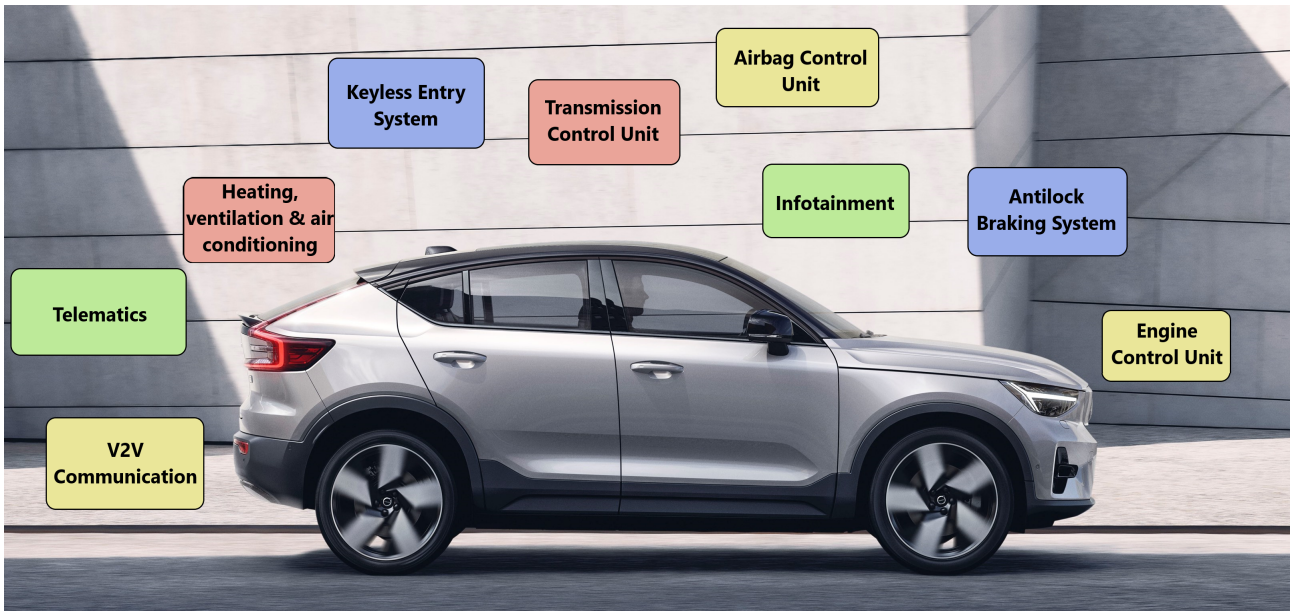

Figure 2.1: Example of IVN and ECU functionality

messaging transfer CAN is using broadcasting technology so that messages without source and destination are broadcast by ECUs in a priority fashion. This means that every ECU is capable to receive any message sent by any other ECU. CAN is a relatively high-speed protocol that for instance is used for the engine, braking and transmission ECUs. The high-speed communication protocols are often used for ECUs that send real-time telemetry data [36]. In addition to high speed protocols there are also lower speed protocols such as LIN. They are often used for simpler tasks e.g. windows, seat belts and locks where there isn't much data to be sent.

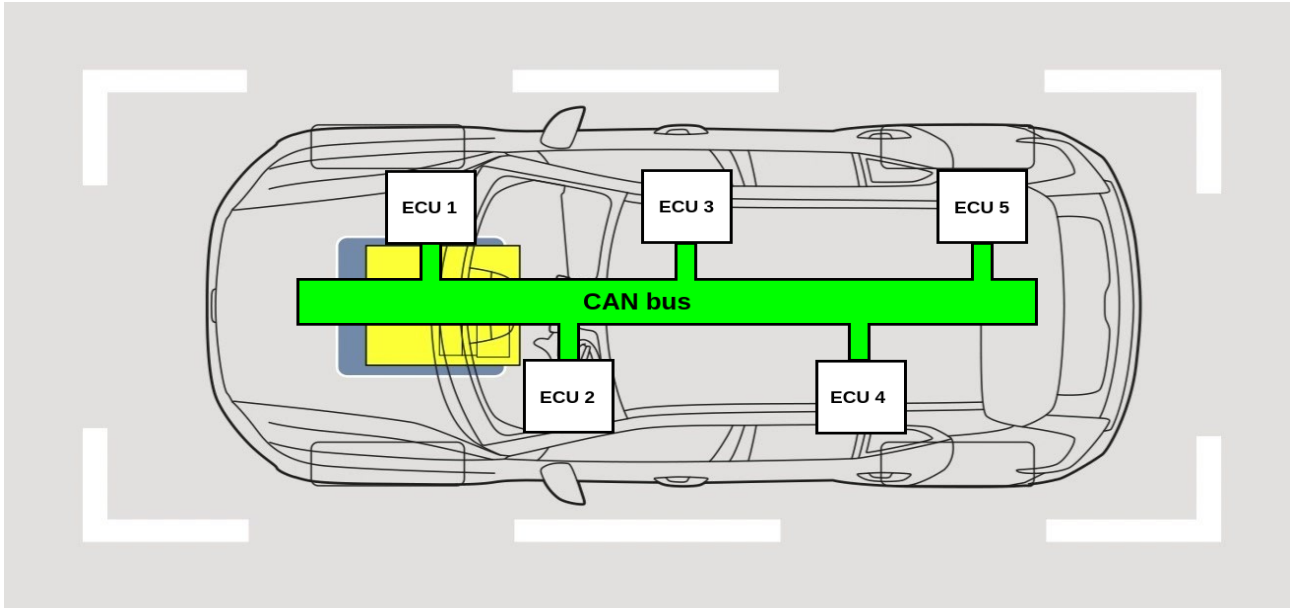

Figure 2.2: CAN bus and ECUs inside the vehicle

In later years, more advanced technology has been put into vehicles and with features such as Advanced Driver Assistance Systems and autonomous driving, there has become a need for higher bandwidth and faster throughput in the communication protocols. For this reason, high-speed protocols like FlexRay and Ethernet have been adopted into modern vehicles to cope with the large amount of data that is produced by e.g. LiDAR, cameras or radar. According to the engineers and technical experts at Volvo Cars that have been interviewed, the current IVN architecture consists of a small number of powerful ECUs, each of which is responsible for many less powerful ECUs. In other words, powerful ECUs create domains (clusters) for which they are responsible. In every such domain, there is a CAN bus with a bandwidth of 0.5 MB/s, meanwhile, the powerful ECUs are connected with FlexRay with a bandwidth of 10MB/s. Figure 2.3 visualises such an architecture.

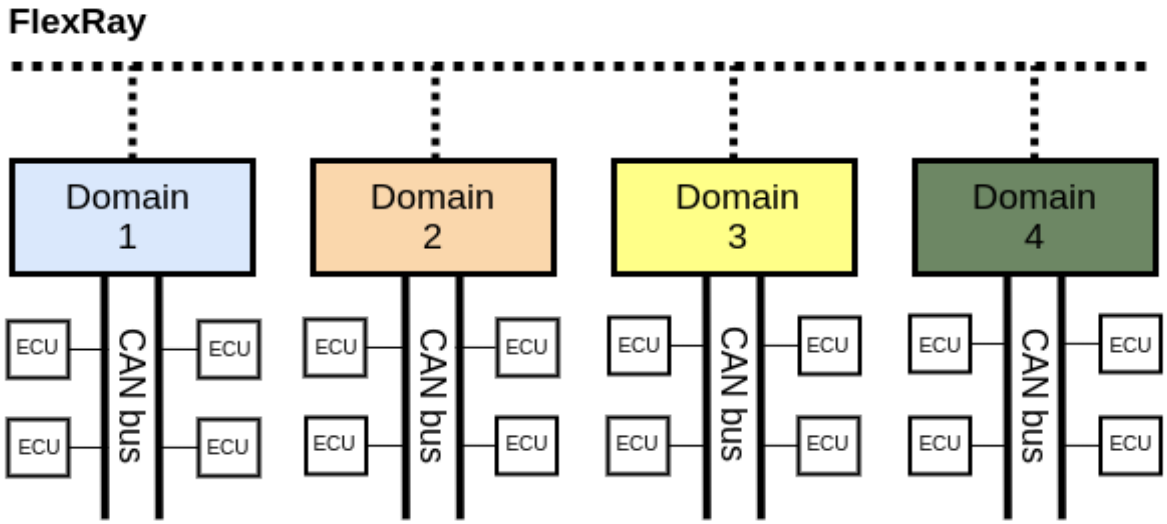

Figure 2.3: IVN network in modern vehicles. Each domain is a powerful ECU which is responsible for a number of simple ECUs with the related tasks

However, this approach will soon be substituted with another architecture that makes use of only one powerful ECU which will connect and control all other sensors inside the vehicle. This architecture still makes use of CAN, but most of the communication is done via Ethernet. Figure 2.4 gives an abstract visualization of the later architecuture approach.

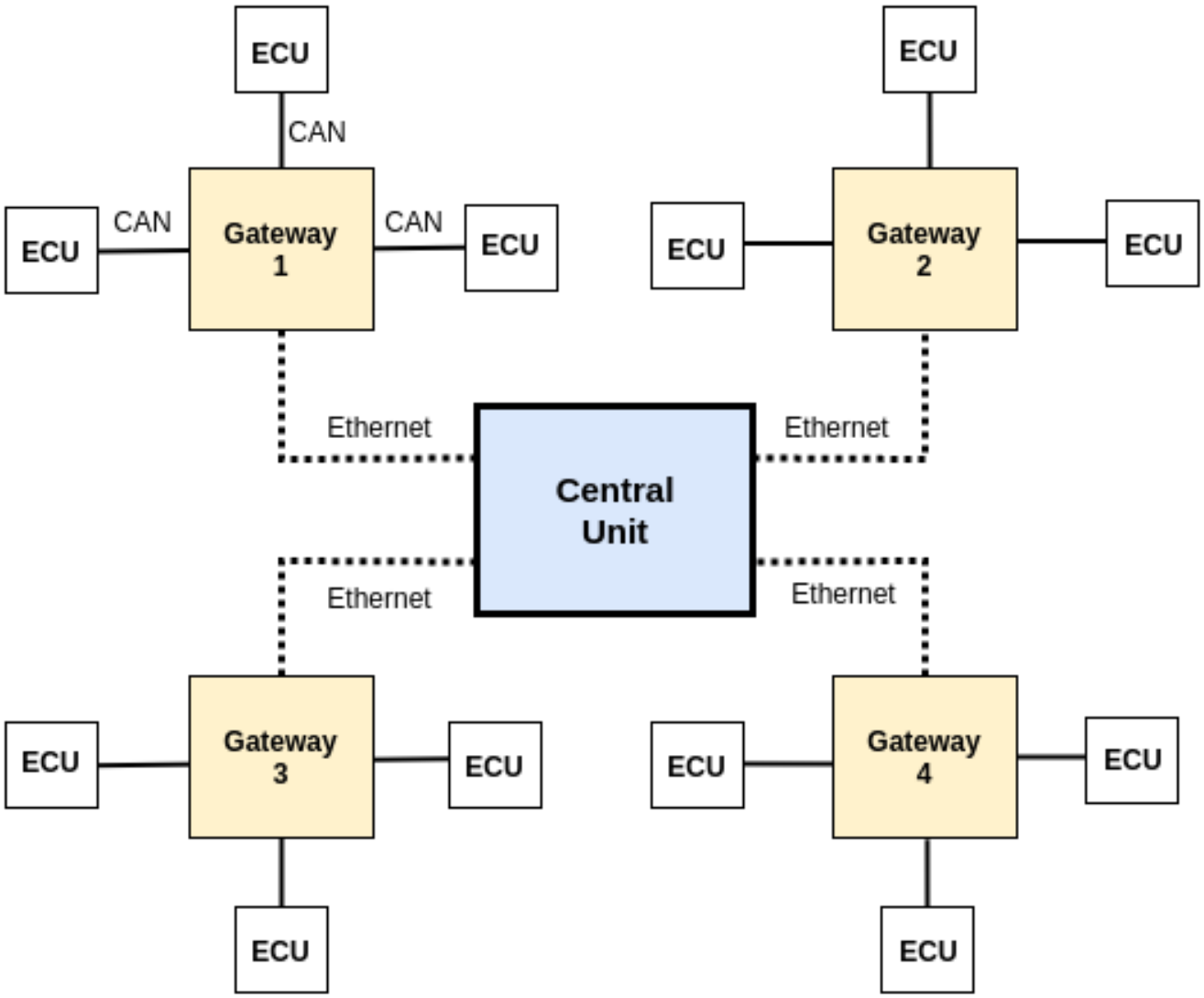

Figure 2.4: Future IVN architecture with a Central Unit.

As can be seen in Figure 2.4, the single powerful ECU is represented as a Central Unit and it is connected to several gateways which unite various simple ECUs depending on their geometrical location. This is a key difference compared to the old architecture which united ECUs in clusters depending on their functionality and not the location. In the new architecture, each gateway is responsible for its area and this allows to have less amount of cables which are shorter because they just need to reach the nearest gateway.

2.2 Security fundamentals

This chapter presents fundamental knowledge in the computer security area that is needed to understand the concepts in this report. Specific cryptographic techniques and terms are explained to provide general knowledge in the area.

2.2.1 CIA - confidentiality, integrity, availability

CIA represents the three core fundamental pillars of information security. *C* stands for *confidentiality* and it emphasizes that information should not be available for unauthorized use. It basically means that data should not be accessible either during storage or transfer by anyone that is not authorized to access it. This is often achieved with some sort of encryption which is explained in the next section. *I* stands for *integrity* and it makes sure that information has not been tampered or modified by any unauthorized person. The integrity of a message or data is often checked either by comparing hashes or checksum which are explained later in this chapter. *A* stands for *availability* and it makes sure that data are accessible whenever it is needed. An attacker can for example try to disrupt the availability of a service by performing some sort of overloading attack [70].

2.2.2 Encryption

Encryption is a method to encode data in a secret way which can be used to achieve confidentiality. It allows only authorized users to read a specific set of data. Usually, the encryption is classified into two types: *Symmetric* and *Asymmetric*.

Symmetric encryption is an encryption approach where parties, participating in the information exchange, only have one and the same secret key that they share with each other. This single key is used for both encryption and decryption. The symmetric key is considered very time efficient because of the usage of only one key, but it also has a disadvantage that all parties need to be in possession of the secret key, which implies that they need to find a secure way of exchanging the key.

Asymmetric encryption implies the existence of a key pair at each party participating in the information exchange: *private* (also known as *secret* key) and *public*. The public key is publicly known to everyone and the receiver's public key is used by the sender for the message encryption. The private key is only known to the receiver who can decrypt the encrypted messages. The private key can also be used as a signature to prove/certify the integrity and origin of a message. Asymmetric encryption eliminates the problem of exchanging the secret key as in symmetric encryption. A disadvantage of asymmetric encryption are the complex mathematical computations and processing of the long keys, which asymmetric encryption is using. This will affect the speed of computation and can not be applied on simple ECUs.

2.2.3 Checksum and Hash

Both hash and checksum are used to determine the integrity of a message. Checksum is a small piece of data which is derived from a bigger data block. If the initial data was modified the checksum over it will be different, thus it helps to determine whether the original data is same or not. Hash is a one-way function that takes an input of a plain text and outputs a hash which is a fixed size string containing what could seem like random data. The thing with a hash function is that it is

deterministic and not random, thus it can be used to verify integrity. This means that the same input value generates the same output hash value and since it is a one-way function there is no way of finding out the input by looking at the output hash. A variant of hash is HMAC and MAC which uses a hash function in combination with a secret key. HMAC and MAC provide both integrity and authentication. The advantages of HMAC are that no encryption is needed to prove the correctness and its high efficiency when it comes to the speed of computation.

2.2.4 Attack types

It is important to understand a number of attack types that can affect the security of the network in the vehicle. The following attacks were considered when trying to eliminate security risks with the attestation protocols.

Eavesdropping attack is a cyberattack that appears when the data, transmitted over the network from one party to another, is stolen by a third unauthorised party (adversary).

Man-in-the-middle is an eavesdropping attack type when an adversary changes and takes control over communication between two parties in a way that the adversary put itself between communicating participants and pretends to act as a legitimate participant for each party engaged in the communication.

Replay attack is a popular form of Man-in-the-middle attack. In this type, the adversary intercepts a message on the communication channel and retransmits it later to the original destination, pretending as a legitimate communication participant. One example can be shown in Figure 2.5, when Alice wants to connect to Bob and Bob asks for Alice's password as a proof of Alice's identity. Alice sends her password (which is hashed), meanwhile the adversary Eve captures this message. After some time, Eve connects to Bob pretending to be Alice, and when Bob asks for Alice's password, Eve sends the hashed password that she captured from the channel. This attack can be prevented by Bob if he sends some random nonce when he asks for the password. To create a random nonce a pseudorandom number generator is needed. A pseudorandom number generator (PNG) is an algorithm that generates seemingly random numbers with the exception that it is deterministic. In this case, Alice will need to hash both password and nonce, so that even if Eve will capture the message and will try to retransmit it later, Bob will send another nonce to Eve which she will not be able to hash together with a captured hashed password.

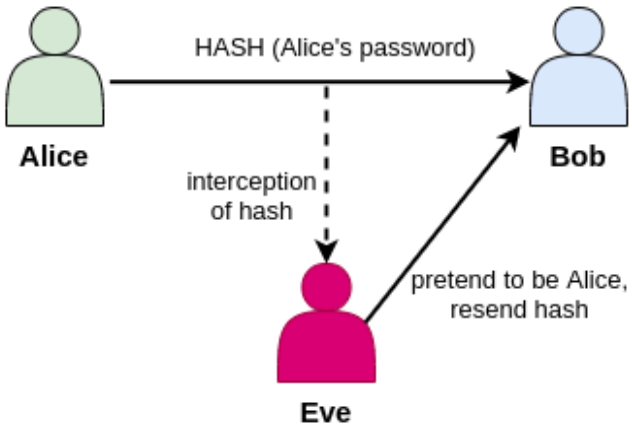

Figure 2.5: Replay Attack. Adversary Eve captures Alice's hash and resends it later to Bob.

Run-time attacks is a type of attack that occurs during the execution of the program. By altering the stack or the heap (memory data structure), the control flow of the software can be changed and for example, a code reuse attack can be performed. Control flow changes are not detected by static attestation methods thus a dynamic method is needed [18]. Code reuse attacks as well as Return Oriented Programming attacks are often performed by changing the return addresses in the stack or the heap to force a change in the instruction order. An adversary can use a control-flow change to either skip certain parts of a software or reuse other parts of a software in a malicious way.

TOCTOU (time of check - time of use) is a cyberattack that is based on the race of conditions when multiple conditions are checked simultaneously, instead they should be checked in a sequence. Basically, the adversary is capable to do unauthorized action by using a small time gap between the time of check and time of use conditions.

DoS/DDoS, Denial-of-Service (DoS) and Distributed Denial of Service (DDoS), are attack types that make the target (user/computer/server) unable to perform the functions that it is supposed to do. This is usually achieved by overwhelming the target with requests and tasks and making the target always busy. The difference between DoS and DDoS is that in the prior case there is only one adversary, meanwhile in the latter one there are multiple parties that try to overwhelm the service, but they all still can be controlled by one single adversary.

Proxy attack is a cyberattack when an adversary is using another computer to perform the attack. This is done in a way that it redirects a communication request from the target to another server/page/computer.

2.3 Device attestation

This section presents the concept of device attestation and how it works.

2.3.1 Basics of device attestation

Device attestation is a security mechanism that verifies the trustworthiness and legitimate state of a device by checking the integrity and authenticity of its hardware, software and firmware. In other words, this technique makes sure that the device has not been compromised and that the code looks as it was initially planned to look so that a third party that is using the given device can rely on it. Usually, the procedure of device attestation is performed before running any process or program that makes use of hardware devices in order to ensure that the devices will work as they are expected to.

A popular form of device attestation is the so-called Remote Device Attestation, which implicates that a device (prover) verifies the trustworthiness and validity of its firmware to a third independent party (verifier), usually in a challenge-response manner. This is achieved in the following way. The verifier sends a unique challenge to the prover. The prover creates a hash of its firmware combined with the challenge and creates a message containing this hash. The prover has a unique key, called the attestation key, which is known only to the prover and the verifier [33], which is used for message encryption. The prover encrypts the message, created in the previous step, and sends it to the verifier. Since the verifier knows the attestation key of the prover device as well as its valid firmware, the verifier can predict what message to expect from the prover. Upon the arrival of the message from the prover, the verifier decrypts the message, compares it with the expected message and determines whether the device is trustworthy or not.

An example of a simple remote attestation protocol can be seen in Figure 2.6. The Verifier sends a challenge C to the Prover, which makes a hash, $hash(C, F)$ over C and its firmware, F. $hash(C, F)$ is then encrypted using the attestation key, which is shared between verifier and prover, and sent to the verifier, which in turn decrypts the message and compares the retrieved hash with the expected hash.

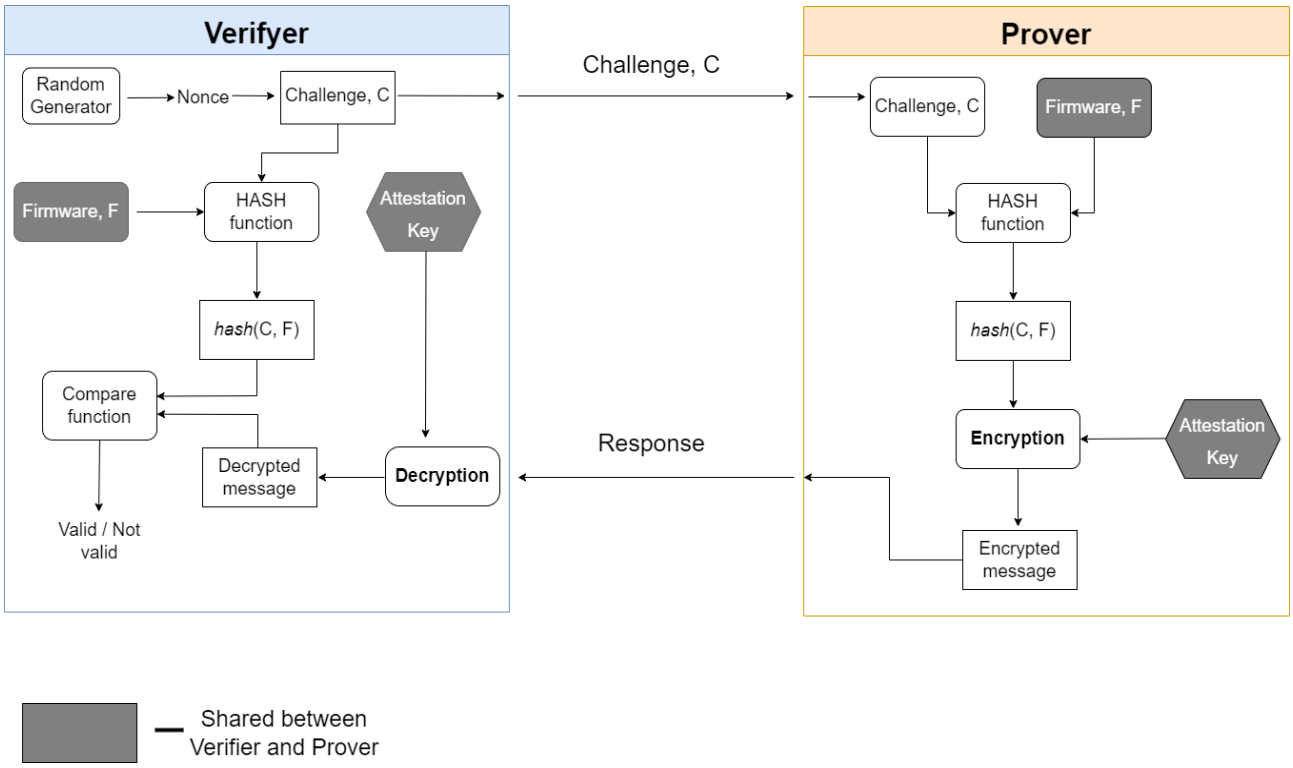

Figure 2.6: Remote Attestation Example

It is worth mentioning that this example is only one of many ways of performing attestation and is demonstrated for the sake of explanation. The attestation can also be done using asymmetric keys so that the Verifier only stores the public key of the Prover. It can also be done by using HMAC. More details on that will come in later paragraphs.

2.3.2 Attestation types

Device attestation is commonly performed by either using software, hardware or a combination of both (hybrid) [3]. Hardware solutions depend on the usage of special built-in secure hardware components such as a trusted platform module (TPM), ARM TrustZone or Intel SGX, which are installed on the prover device. An example of hardware attestation can be to calculate an HMAC over the prover's firmware, which implies that the prover needs to have a secret key shared with the verifier which only is possible by using secure hardware as storage [47].

Pure software-based solutions often rely strictly on computational times and therefore differ a lot from what we know as cryptographically secure [12][54]. Cryptographic keys are not used at all, instead these solutions rely on the assumption that a malicious software can't perform the correct calculations in the expected time frame.

Lately, researchers have tried to combine software and hardware attestation to make them more suitable for the vehicle and other industries. The combination of these

two approaches is called Hybrid attestation. Researchers have tried to find ways of keeping the security features of cryptographic keys when it comes to authentication and integrity, but at the same time not be dependent on expensive hardware solutions [33][34][48][3][42][50]. The primary way that this has been addressed is by coming up with solutions that provide secure storage of cryptographic keys, similar to the features of a TPM but without the expenses that it brings. Some methods that have been developed for this purpose are Physical Unclonable Function (PUF), Key predistribution system (KPS) and Protected bootloader (Flash Protection Regions), which is described later.

2.3.3 Physical unclonable function

Physical unclonable function, or PUF, is a function that works as a hardware fingerprint (device's unique structure) and can be used to generate cryptographically secure information. How this can be achieved is by leveraging the uniqueness and unclonability of the silicon inside the SRAM chip. SRAM is a relatively common chip on low-end embedded devices but it does not always exist. The way PUF works in practice is by using a challenge-response approach [3]. For each challenge, a unique response is created and since PUF is deterministic it always returns the same response for a specific challenge. It is also highly efficient with an approximated time consumption of only 0.4 μs [3].

When using PUF as secure storage of cryptographic keys, the verifier stores challenge-response pairs (CRP) during a setup phase between the verifier and the prover. As can be seen in Figure 2.7, the verifier sends a number of randomly generated challenges to the prover which runs the challenges through the PUF and then sends back the responses. The verifier then stores these CRPs in a secure database. The verifier can then use the CRPs later to authenticate the prover.

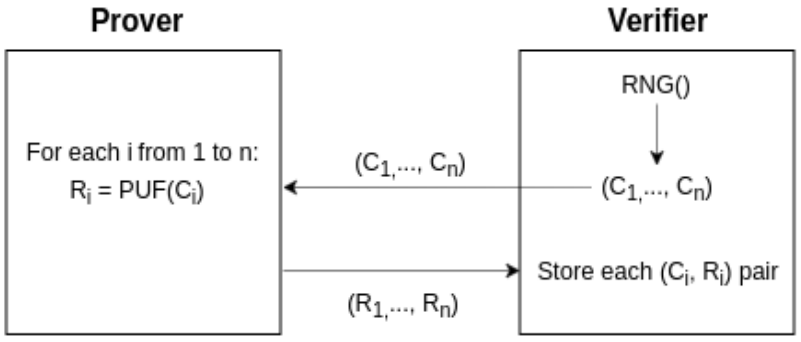

Figure 2.7: PUF setup phase. RNG is random generator, $C_{1...n}$ are challenges produced by random genrator, R is responce made by PUF.

Studies have shown that some PUFs can be unpredictable under certain operating conditions such as temperature variation. In [28] and [52] experiments were conducted and showed that e.g. Flip-Flop memory is not suitable for PUF since the

error (unpredictability) significantly increases during temperature change. Further, it showed that SRAM doesn't have the same problem, thus it is suitable for PUF.

2.3.4 Key predistribution system

Key predistribution system, or KPS, makes use of a predistributed algorithm for unique symmetric keys generation between all ECUs [48][42][50]. It works in the way that one center (either a specific ECU or Car manufacturer) creates a bivariate polynomial and then distributes a unique key generation algorithm $F_i(j)$ to each ECU, where i is the sender ID and j is the receiver ID, so that $F_i(j) = F_j(i)$.

Experiments have also shown that KPS is a relatively time-efficient approach compared to asymmetric encryption methods such as RSA [42][48]. In [48] it is explained that the computation time of RSA2048 is approximately 32,265 μs while 358 μs for KPS, which makes KPS clearly faster than RSA, but, compared to PUF, KPS is significantly slower.

2.3.5 Protected bootloader

Protected bootloaders are becoming more common in low-end embedded devices inside the Read-Only Memory (ROM). They also come with three strong security measures, read protection, write protection and uninterruptibility [33][34]. The last measurement can be achieved by using the so-called Interrupt Vector Table (IVT), a table with interrupt requests and the respective interrupt handlers.

Also, modern devices can be equipped with execute-only memory (XOM). If a protected bootloader section exists, cryptographic keys can be stored there together with the attestation software. This section is then protected by a lock bit which prevents outside code to read this memory section, thus only allowing the attestation code to access the secret information. The advantage of performing the attestation at device start-up in a protected bootloader is that it is done before any malicious code has the chance of executing.

Generally, the protection of the bootloader is done by something called flash protection. Flash protection cannot be undone without physical access to the device, and even if an adversary gains physical access there is no way of removing the *flash protection* without erasing the memory region. Flash memory usually has two parts: bootloader section (BLS) and application section [33] and for each part, a lock bit for reading, writing or preventing the interruption can be set.

2.4 Requirements

The requirements for embedded systems inside the IVN differ from ordinary embedded systems. Some special requirements that are specific for embedded systems in vehicles are scalability, performance, robustness and security. It is also important to

have in mind that the automotive industry is controlled by cost and always strives for low-cost solutions.

2.4.1 Attestation requirements

To ensure that an attestation method is secure, security requirements are needed. These are requirements that an attestation method needs to fulfill in order to be classified as secure. Keep in mind that requirements can vary a bit between different attestation methods depending on their characteristics.

One situation where attestation needs to be performed can for example be right after the vehicle is unlocked and before the vehicle starts moving. This implicates that the time frame for performing the attestation must be very short, within a few seconds to not make the driver wait when it needs to drive. Therefore, we can conclude that attestation must be done during a limited time constraint.

After looking at a number of papers, the following requirements have been found for performing secure attestation:

- **Exclusive Access to Secret key**: Attestation keys are used to verify the integrity of the prover. To keep this key secret and only available to the attestation software is crucial, but also one of the hardest requirements to satisfy for low-end embedded devices [19].
- **Immutability**: To ensure that the attestation software cannot be modified from its original state is crucial. This is often assured by placing the attestation software in ROM [23].
- **Uninterruptability**: It is crucial that an adversary can't interrupt the attestation program during execution. If an interruption would arise, the adversary would be able to extract secret information from the memory [19][34].
- **Unforgeability**: It is important that the identity of a prover can't be forged. A valid signature should for example not be able to be produced by an adversary [22]. It is also necessary to resist, replay, cloning and impersonation attacks.
- **No Leaks**: To ensure that the adversary can't learn anything about secret information this requirement is needed. To satisfy this, all information that potentially could leak secret information should be erased from memory after execution or when not needed.

2.4.2 Vehicular requirements on security and attestation

The cybersecurity management system was declared in the United Nations Regulation 155 (UNR155 or R155) and adopted in 2021 [64]. UNR155 implies ensuring cyber security during the vehicle's life cycle and minimizing security risks [65]. Moreover, this obligates creating Cyber Security Management System (CSMS), its documentation and performing security audits for any vehicular type. UNR155 will be obligatory for all vehicular producers in the EU from July 2022 [63]. Standard ISO21434 describes the framework for establishing security in the vehicles and helps

to cover the security criteria specified in R155 [14].

The authors in [48] discuss a number of requirements for device attestation in vehicles. According to them, the key to establishing a secure environment in the vehicle is to have long-term security, since the vehicle is usually used for at least 10-15 years. Secondly, the authors state that some of the critical ECUs must have a quick response time, less than 1ms. Thirdly, the authors propose the following points to be fulfilled when it comes to security architecture:

- Every ECU can only establish communication with an ECU that has been approved and certified by the vehicle's manufacturer.
- Each ECU must be able to prove its authenticity as well as the integrity and be able to perform the encryption of the messages sent to another ECU.
- The control procedure over the authentication and integrity must be replaceable if the software has been updated.

Studnia et al. in [59] stated that to have a secure inter-ECU communication in CAN the following properties must be followed:

- **Confidentiality**: a message sent on CAN is broadcast to every ECU, therefore it should be protected from reading by any party not participating in communications, as well as it should be protected from eavesdropping.
- **Authenticity**: a device needs to be able to identify the actual origin of the message which is problematic for CAN messages since they do not have a dedicated field of their source destination.
- **Availability**: All the devices will work properly and provide the service they are expected to provide.
- **Integrity**: the command given to the device should be unchangeable during its transmission.
- **Non-repudiation**: there should be no possibility for ECU to deny the sending or receiving message if the ECU has sent or received respectively the message.

2.5 Adversary model

There are two levels when it comes to the adversary model of this project. One model is stricter than the other, but both are taken into account and discussed in the project. The first and less strict adversary model assumes that an adversary only has remote access to the IVN. In this model, no hardware features can be changed or added to the IVN. In this case, a device attestation protocol should at least be able to protect against remote attacks since physical attacks are harder to carry through and less scalable.

The second model is more strict than the prior one and assumes that an adversary has both remote access and physical access to the IVN. This implies that the adversary can tamper with the software on all ECUs and listen to/modify the communication inside the IVN. It also means that hardware components can be added/switched/removed physically from the vehicle.

3
Method

This chapter presents the method and approach which were applied during the project to achieve the above-mentioned goals, described in Section 1.

3.1 Literature review

In total two methods were used to find relevant articles in the area of device attestation. The first method was to search for articles on Google Scholar. During this method, we used a set of keywords for a structured approach to finding relevant articles. An article was seen as relevant if it mentioned "attestation" in either the abstract or conclusion.

The following keywords were used:

- Device attestation
- Device attestation vehicle
- Embedded device attestation
- Attestation vehicle
- Remote device attestation vehicle
- Software attestation
- Hardware attestation
- Attestation IoT

The second technique that was used is called *snowballing*. This is done on the articles that were found during the first step. Snowballing means reading the reference list of the previous work in the area and then checking if they are relevant in the same way as was done in step one. All relevant articles from steps one and two were added to an excel document where we also added suitable categories to classify different articles.

The reading process was divided into three phases. This was done to make it easier to digest a number of papers at the beginning of the literature review and to later move on to reading more and more in-depth for a broader understanding.

The three phases are:

- **Phase 1**: In this first phase the papers are read in a brief way. This only includes the abstract and the conclusion. During this step, the goal is to understand if the paper is relevant to attestation or not.

- **Phase 2**: In this phase, the introduction and some parts of the theory/background are read. Phase 2 is only applied to the relevant papers from phase 1. In this phase, we also put a ranking on each paper to be used later in phase 3.
- **Phase 3**: Finally, the most relevant papers based on ranking are read thoroughly to get a deep understanding and to later be used in the final report.

In total, 112 articles have been reviewed, but not all of these articles were included in the final report. The decision of either excluding or including an article was mainly based on the ranking from 1-10 that it was given in the litterateur review. Articles were prioritized from high to low ranking. All articles with ranking 8, 9, and 10 were included in the report. When it comes to ranking 6 and 7, there were some better than others, thus the ones that were most relevant to device attestation in low-end systems were chosen.

3.2 Proposed attestation methods

Our own proposed attestation methods in Section 5.2 were based on state-of-the-art attestation protocols which was discovered in the Literature review results (Section 4). During the designing phase of the protocols, today's vehicle-specific requirements were taken into account in order to make the protocols applicable to the modern vehicle industry. After the protocols were designed and described, a number of industry experts from Volvo Cars and Volvo Trucks were interviewed and asked to analyze the proposed solutions. Their opinions and thoughts were taken into consideration and described in Section 5.3.

3.3 Limitations

A project in this area can be significantly larger than this one, but since we had a limited time frame, we had to narrow down our scope and focus on fewer tasks. One of the main constraints has been the time since this is a master thesis which should be written in a period of approximately 20 weeks. The literature review has always been the primary objective, thus where the main focus should be, but at the beginning of the project, we also wanted to implement an attestation protocol, if time would have let us. In the end, the time-limiting factor made us focus on the literature review and leave out the implementation and experiment part.

4

Literature review results

This section presents the result of the literature review and mainly describes different approaches to device attestation. The attestation methods are categorized into the following categories: Master ECU Approach, Decentralized Attestation Approach, External Attestation Approach, Hardware Attestation Approach, Software Attestation Approach, Hybrid Attestation Approach and Dynamic Attestation Approach. The methods are not necessarily tied to only one specific category and can overlap.

4.1 Master ECU approach

The Master ECU approach refers to the type of attestation where there is one main ECU, *master*, which performs the attestation on all other ECUs in the vehicle. This is one of the most used device attestation approaches and has multiple types of implementations. There are different approaches of Master ECU attestation presented in [33], [48], [34], [3], [50] and [42].

4.1.1 Internal master ECU approach

By internal Master ECU attestation means that the attestation makes use of a specific device (master ECU) for attesting other ECUs and all processes happen inside the vehicle. One of the protocols was proposed in [33] and it is called ALE. This protocol follows some general assumptions. For every device to be attested, the protocol requires:

1. Existence of an immutable bootloader (write-protected part of memory that is executed from the very start)
2. A unique attestation key ak which can only be accessed by the immutable bootloader
3. Uninterruptible execution of code in the immutable bootloader

To achieve the first property, the author in [33] proposes to enable a lock bit at the bootloader section of the device's flash memory (B) for reading and writing to prevent interruptions. Alternatively, the device can be protected with XOM (if this is available) or ROM, and, in this case, B can represent this memory region. As for the second property, the author proposes to place ak inside of the bootloader section and set a lock for any read access attempts that are performed by a piece of code located "outside of the separated memory region". If setting such a lock is not possible, the author proposes to store ak in a secure place in such a way

that the access to it "can intentionally be denied until the next device restart". In other words, ak is accessible for B only at the start of the device process. Finally, enabling the property of uninterruptible execution can also be done via a special lock bit in the bootloader section which is a part of flash memory. According to the author, this can also be achieved by storing the interrupt vector table (IVT). The IVT can perform the check if the interrupt was triggered during the execution of B and if this the case the interrupt is denied. Otherwise, the interrupt is sent over to "user-defined interrupt handler" which will process it.

Protocol ALE consists of two phases: *deployment* (each device is initiated by the network operator O) and *attestation* (the software on each device D_i is verified). In the first phase, O deploys device D_i with its unique ak (which is stored at B), code for interaction with O, unique identifier i and boot nonce N_b (all three are stored in mutable memory outside of B). Additionally, O stores information about each initialized device by storing the device's ID, ak and a valid firmware version. When both O and the devices have been deployed with the necessary information, the attestation phase can be enabled. Also, the author divides ECUs into two categories: **safety-critical** ECUs (which are attested as soon as the vehicle's door is being unlocked) and **non-safety-critical** ECUs (which can be verified later on). The list of the prior type can be identified by the vehicle's manufacturer.

The attestation phase works as follows. Each ECU stores its attestation key, AK, and nonce variable, N_b, in the secure protected storage. As soon as the vehicle is unlocked, each ECU boots and computes its integrity by taking the hash of its software, and stores the hash in the IM variable. Then, the ECU generates its response key for the master request, RK, by taking the HMAC over IM and N_b with its AK. The master ECU generates a random nonce, N, and sends it to all safety-critical ECUs first. Each safety-critical ECU receives N and overwrites its N_b with N which leads to a modification of RK in the next round. Afterwards, the ECU sends a response to the master ECU which consists of the ECU's ID and HMAC, σ, over newly arrived N and ID with the first version of RK (counted before the arrival of N). Finally, the master verifies the arrived response by computing the expected response and comparing it with the arrived response. Since the master ECU stores each ECU's IM and AK, it can predict what response to receive from each ECU.

Figure 4.1 visualizes how the ALE protocol works:

1. The Simple ECU boots and creates a hash over its firmware, then the derived response key is calculated;
2. The Master ECU creates a nonce and sends it to the Simple ECU;
3. The Simple ECU receives the nonce, overwrites N_b and creates an HMAC of the received nonce, response key and ID;
4. The Simple ECU sends σ (HMAC from the previous step) and ID to the Master ECU and the Master ECU verifies the ECU by comparing the received and expected values;

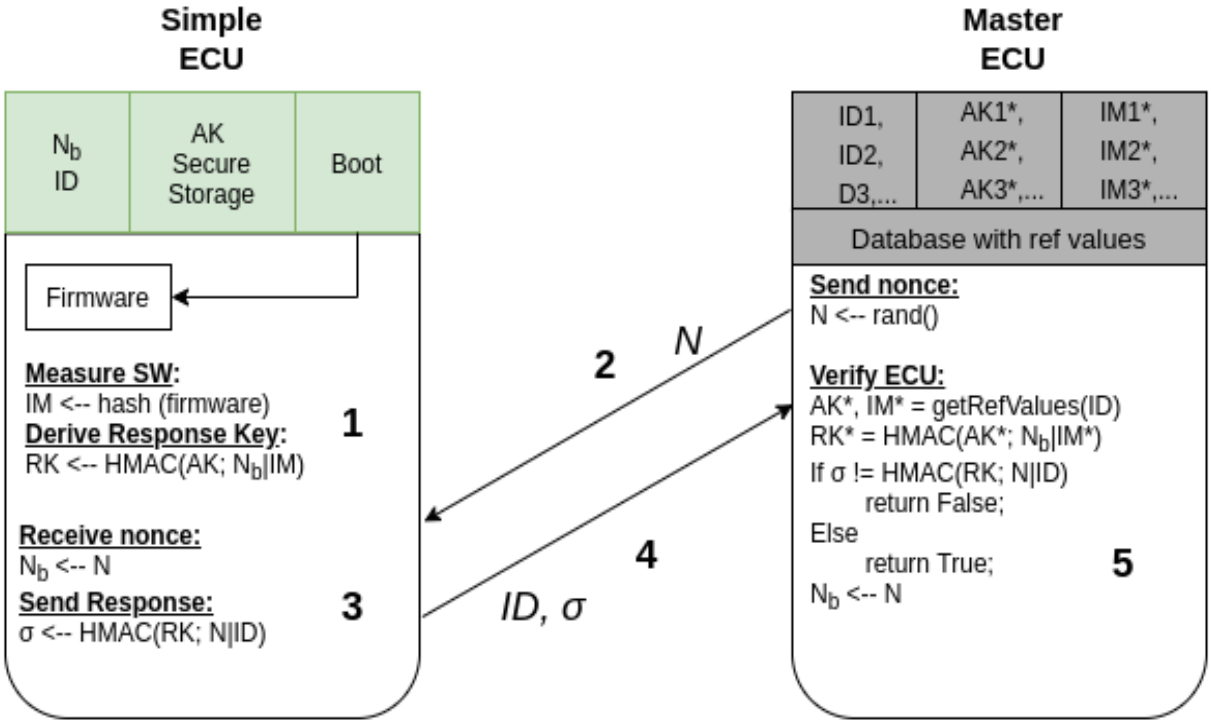

Figure 4.1: ALE protocol - Master Attestation Protocol

4.1.2 External master ECU approach

By External Master ECU attestation it is meant that there is a usage of a master device, but at the same time the vehicle establishes communication with an external device (e.g. remote server) to perform the attestation. An attestation scheme that uses this attestation type was proposed in [48]. The main idea is to use a single master ECU, similar to the one that was described in the previous Section 4.1.1, but the verification by itself is performed in a remote server, which the vehicle needs to establish a connection with. How to establish a connection with the remote server is described later in Section 4.3, to describe this model we assume that such a connection is established. Figure 4.2 shows the architecture of the internal vehicle environment needed to enable this protocol. The proposed scheme has multiple simple ECUs, one master ECU, one communication ECU and one remote server called Center. The communication ECU acts as a bridge between each ECU and the Center. The Center contains information about valid software versions and their ROM hash values, symmetric keys to establish the connection to other ECUs and KPS key generation matrix which is randomly generated for every vehicle. The master ECU's main task is to verify simple ECUs, it contains a symmetric key to establish a connection with the center, a list of acceptable hash values and a KPS key generation algorithm. Each simple ECU contains a shared key with the Center to be able to establish a secure connection and a KPS key generation algorithm. Both the master and simple ECU have their own serial numbers (ID).

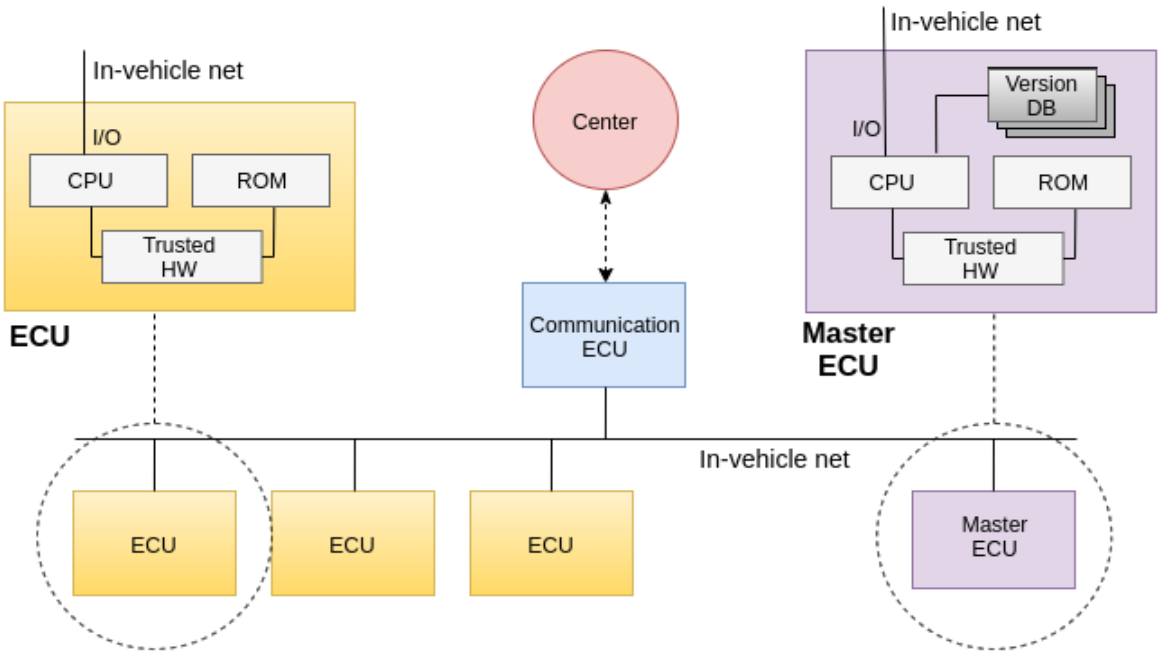

Figure 4.2: Internal vehicle architecture for the protocol using master and remote Center

Figure 4.3 visualises and formally describes the protocol. Steps 1.1-1.4 represent the initialization phase between the ECUs and the Center, this is when keys are delivered to each ECU. The blue color represents steps at the initialization phase between the master ECU and the Center, meanwhile red color represents steps between the simple ECU and the Center. The green color represents attestation protocol between the master and the simple ECUs and secure communication between simple ECUs.

Now we describe in more detail how the protocol works. The master initiates communication by sending its serial number (E_m) to the center (step 1.1 blue in the Figure 4.3). The Center retrieves the master's key from the serial number, K_m, and generates a random number, R_m, which will be used as a session key and sends it to the master with the master's key K_m (step 1.2 blue). The master ECU performs attestation of its hardware by hashing its ROM ($H(ROM_m)$) and sends it together with the vehicle number, V, to the Center with the key received from the center in the previous step (step 1.3 blue). Afterwards, the Center verifies the master's hash by comparing it to the hash in the list of valid hashes of the master ECU, generates KPS matrix A_{ij}, and sends the key generation algorithm $F_m(y)$ (for calculating symmetric keys) and the list of valid versions and hashes for each ECU pair in the vehicle (step 1.4 blue). The communication between the Center and the simple ECUs is done exactly in the same way (steps 1.1-1.4 red) with only one difference at the last step: the Center sends the key generation algorithm and the correct hash of the master ECU's ROM.

The next phase of the protocol starts when the driver starts the vehicle. The master ECU generates two random numbers (r_1, r_2) and broadcasts the prior one (r_1) to all simple ECUs (step 2). Upon the arrival of this message, the simple ECU retrieves r_1 and takes a hash over its software. Then, the simple ECU sends a message, $Sig_{F_x(E_m)}\{E_x, H(ROM_x), r_1, r\}$, where $Sig_{F_x(E_m)}$ denotes "a symmetric-key message-signature pair using a keyed-hash function such as in HMAC" and r is a challenge for the master (step 3). The master, in turn, uses the key generation function $F_m(E_x)$ together with the measured hash from the ECU to validate the correctness of the signature of the message by comparing it to the hash in the list of valid hashes. If it is correct, the master sends the hash of its ROM together with the received challenge and number r_2 to the simple ECU: $r, r_2 H(ROM_m)_{F_m E_x}$ (step 4). Finally, the communication between two ECUs $(E_x -- > E_y)$ can be done by using following format of the messages: $(payload, r_2, counter)_{Fx(y)}$ (step 5). The variable *counter* is used in this case to prevent a replay attack. The protocol assumes that each ECU has one sending counter (which is incremented every time the message is sent) and a set of receiving counters in the form of a function $counter(x)$ where x is the ID of any other device from whom the ECU received the message. In other words, every time the ECU receives a message from a device E_x with counter c, the ECU performs the validity of condition $counter(x) < c$, and after successful verification, $counter(x)$ gets value of c.

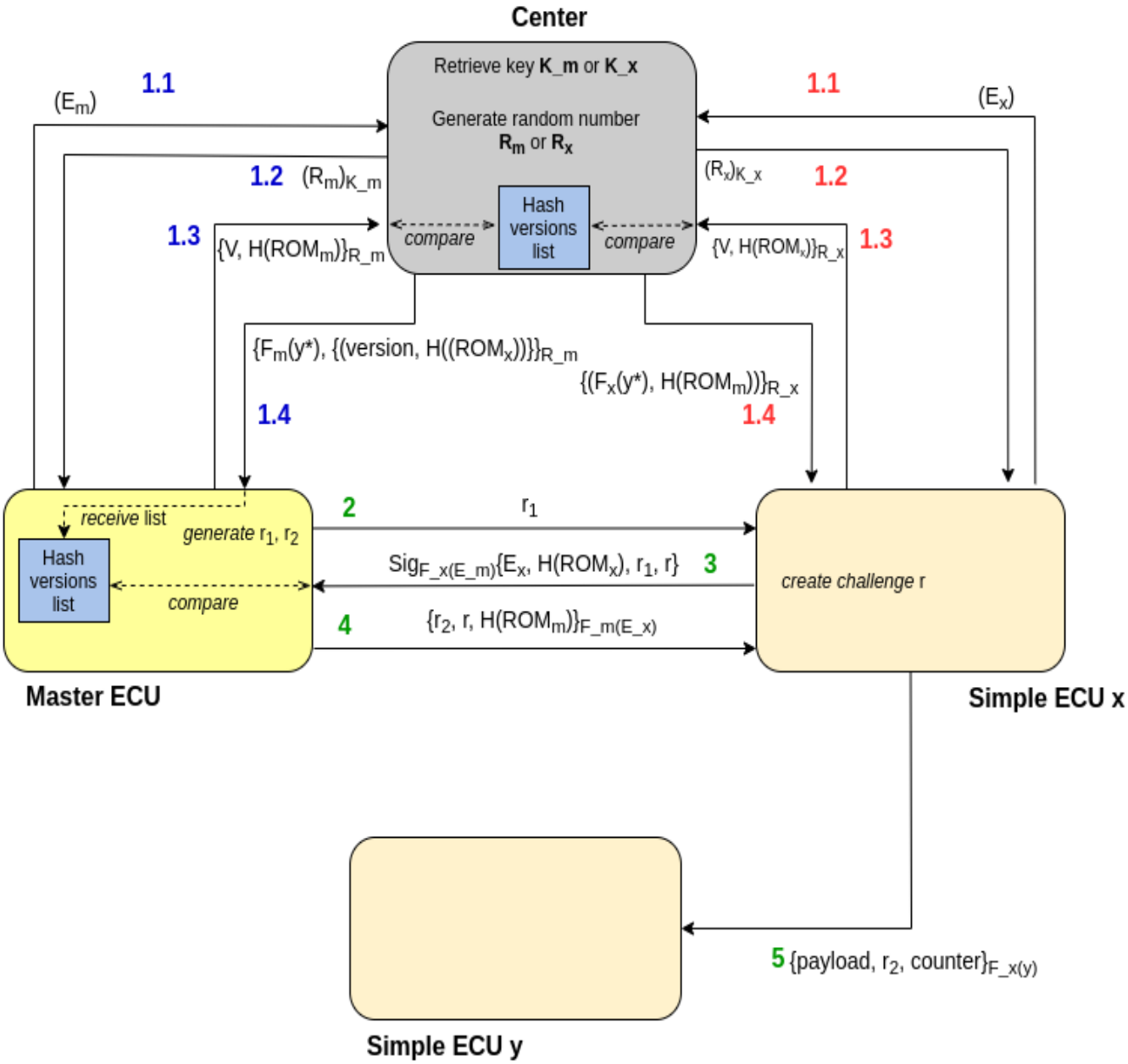

Figure 4.3: Master ECU approach using a remote verification Center.

4.2 Decentralized attestation approach

The attestation methods that have been explained in Section 4.1 are all centralized approaches. A centralized approach has one disadvantage: a single point of failure which could make life easier for an adversary. Instead there exist decentralized attestation methods where all ECUs in a vehicle can verify each other's firmware. This makes it much harder for an adversary, since a significantly larger amount of ECUs needs to be compromised for the attestation to fail.

In the paper [29], an example of a decentralized attestation protocol is given. To start with, this protocol requires secure hardware at each ECU to store cryptographic keys and the hashed firmware of other ECUs in a database. For security this protocol uses the Double Ratchet (see more details in [29]) algorithm to create symmetric keys. New keys are generated for each message thus resisting replay attacks.

In Figure 4.4, the protocol is shown. For time efficiency, not all ECUs are attesting to each other. The mutual attestation is performed by those ECUs which are co-reliable and codependent. An example of ECUs that rely on each other is the ones that handle driving assistants, cameras and LiDAR. Such a group of ECUs is called *attestation group*. The first step of the protocol is that each ECU acts as a challenger and sends out an attestation request to all ECUs within its *attestation group*. In the second step, each prover receives the attestation request and responds with its hashed firmware in combination with a MAC. In the last step, the challenger compares the response with the expected firmware hash that it got stored in a secure database.

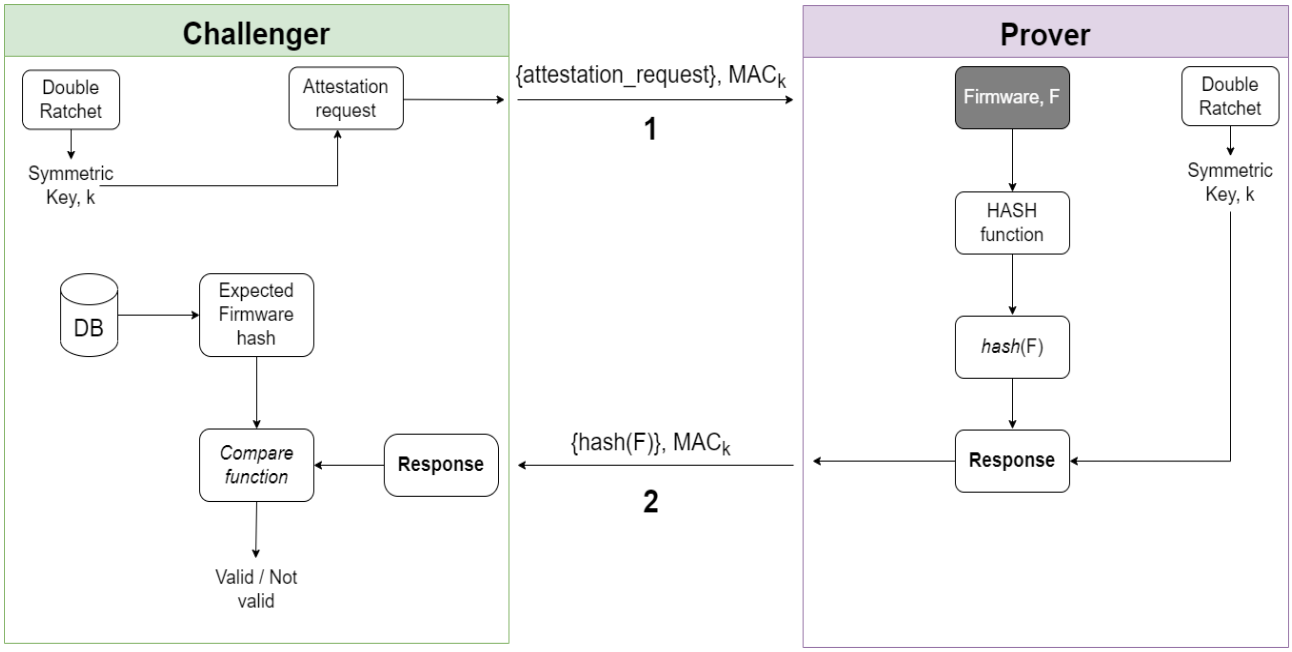

Figure 4.4: Decentralized Attestation Approach

This protocol can then be performed either in a sequential or parallel manner. According to the conducted experiments in [29], the sequential approach is significantly slower than the parallel. For 100 ECUs it would take approximately 180 seconds to attest them sequentially and only 3.5 seconds if done in parallel.

One approach in decentralized attestation by using a spanning tree was proposed in [33], called SALAD. This collective attestation protocol is based on a method where

one special node sends a verification request to an arbitrary neighboring node, which in turn initiates the attestation of itself and saves the result to a so-called *verification report*, which consists of all devices that have been verified by the node. Then, this node communicates with its neighboring nodes, by sending an attestation request and asking for their verification reports. So the node collects the reports from all its neighbors, verifies them and adds new verified nodes to its report. Later on, the neighbors do the same procedure with their neighbors by sending a request and exchange the reports. As this continues as a chain process, each node in the network eventually has a verification report of all nodes in the network.

For a detailed description of the protocol we refer to the main article [33], in this section we provide a light version of the protocol description. Initially, SALAD performs the Deployment phase when all devices in the network are initialized and equipped with all necessary tools for performing the attestation. Following this, the protocol performs the Attestation phase which consists in turn of four additional phases: Initiation, Generation of Attestation reports, Distributed report aggregation and Finalization. At the Initiation phase, a special node O (network operator) sends to an arbitrary device D_i an initial message msg_{init} consisting of:

- a set of valid software versions in the network (vss);
- current attestation phase pid;
- a signature sig over vss and pid by using its secret key $sig = (vss, pid)_{sk}$;

Further, at the Generation of Attestation reports phase, the device D_i receives the message msg_{init}, verifies its signature with the public key of the O and checks if the phase is new by looking at the pid. If it is the case, then D_i verifies its software, sets its current phase id $currpid$ to pid and generates its attestation report rep. The attestation report consists of $rep.ids$, which lists all valid devices that D_i got verification on, and $rep.proof$ which provides proof of the validity of the devices in $rep.ids$. From the beginning, the proof consists of D_i's id and MAC over its pid with D_i's attestation key ak_i. Then D_i sends an announcement message msg_{ann} to the neighbouring node(s) D_k consisting of its $currpid$ and $rep.ids$ (which is only i).

In the fourth phase, Distributed Report Aggregation, D_k checks the freshness of $currpid$ and in case D_k's $currpid$ is smaller than the received one, then D_k will send a message to D_i to request msg_{init} from O. If the $currpid$ is fresh, then both devices exchange their attestation reports. D_k will send an exchange request only in case D_k has at least one verified device less in the D_i's report (this check is done by comparing $rep.ids$ and the D_k). If the difference is one or more, then D_k sends a request msg_{repreq} to D_i, which in turn generates a subset of attestation proofs, $srep$, of the devices that D_k lacks. $srep$ is authenticated with MAC mac_{ik} whose key is taken from channel-key ck_{ik} (shared between D_k and D_i) and $currpid_i$. D_i sends to D_k a response message msg_{rep} which consists of $srep$ and $macik$. D_k receives the message, recomputes $macik$ which checks the integrity and authenticity of $srep$. In case of successful verification, D_k adds $srep$ to its report rep_k and generates its own msg_{ann} to its neighbours. At the last phase of Finalization, O connects to the arbitrary device D_i, asks for the attestation report, checks the presence of all devices' validity and sends a stop message to stop the attestation procedure. Fig-

ure 4.5 demonstrates different stages in the SALAD protocol, which have just been described.

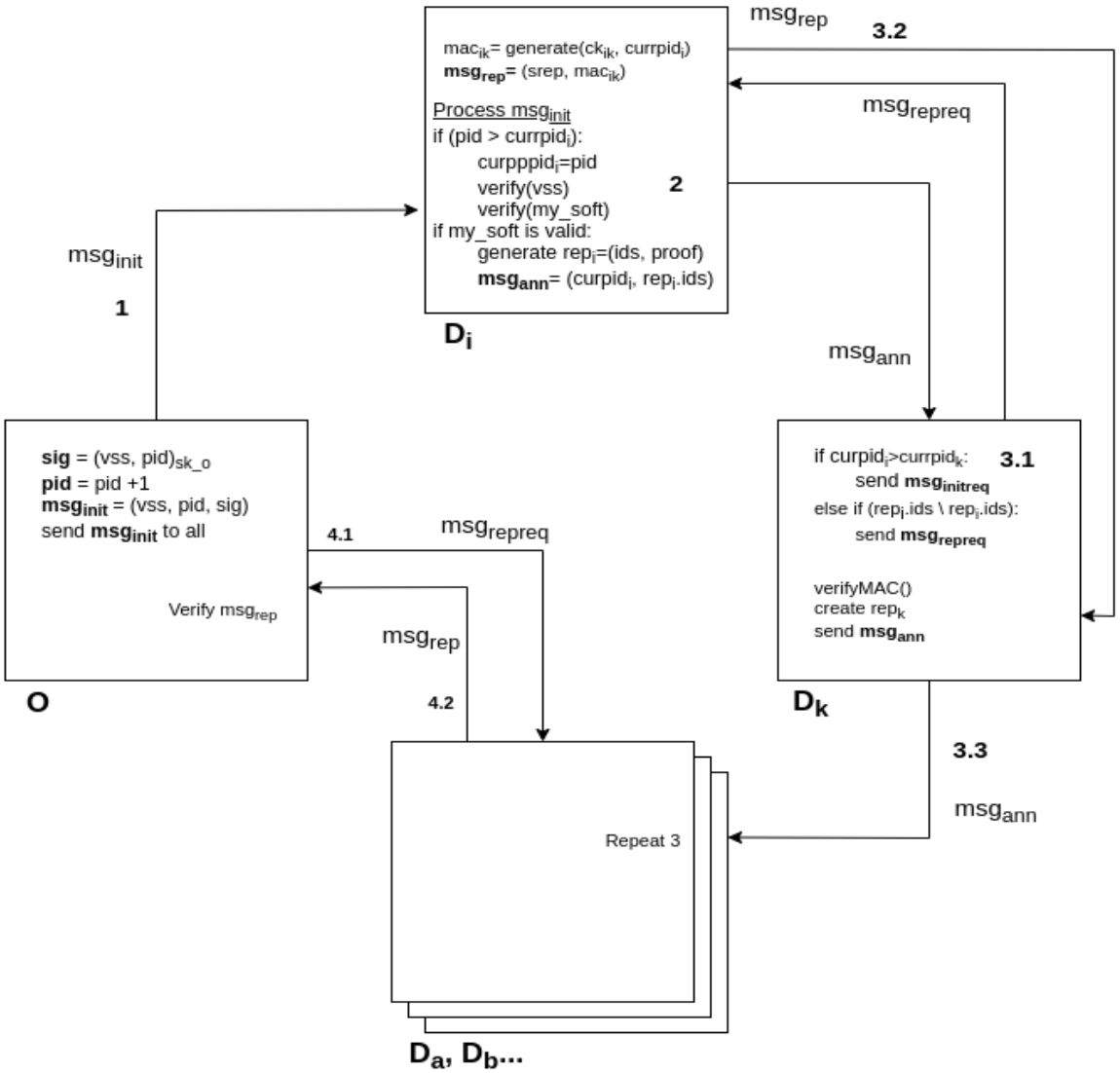

Figure 4.5: Salad protocol. 1. Initiation, 2. Generation of Attestation Reports, 3. Distributed Report Aggregation, 4. Attestation Finalization.

In [33], a decentralized attestation method called PASTA is suggested. The method leverages the Schnorr multi-signature scheme to verify the integrity and validity of each ECU. Further, PASTA depends on secure hardware on each ECU where public keys and symmetric keys to all other ECUs are stored. The protocol itself must also be stored and executed inside the secure hardware so that an adversary can't tamper with the protocol or read secrets without physical access. There also exists other similar methods such as SANA[7], SEDA[13], SlimIoT[9], US-AID[25], $SAFE^d$[66] and LISA[15] which are so-called swarm/decentralized attestation methods.

PASTA works as follows:

- Each ECU periodically acts as an initiator by verifying its own software and measures the time since the last token was generated. If the software is valid and the time since the last token exceeds a predefined threshold, the initiator sends out a token generation request to all its healthy neighboring ECUs. A token contains the following parameters T.ts=timestamp, T.ids=ECU ids, T.sig=Schnorr signature, T.valid=validity of token.
- The receiver ECUs check whether the T.ts is fresh, that the ECU from which it received the token request is healthy and if its own software is valid. If this is the case it forwards the token request to all its healthy neighbors. This can be seen as a chain of trust.
- The token request is then sent back through all the ECUs to the initiator. On the way back all healthy ECUs append their ID to T.ids.
- When the initiator has received all the responses it creates a challenge. The challenge is then sent out as in the first step. Each ECU partly signs the challenge and sends it back to the initiator.
- The initiator aggregates and verifies the signature, if it is valid, the token is stored in the ECU and distributed to all other ECUs in the network. A valid token shows which ECUs are healthy and at what time it was determined. The stored tokens are used to determine which ECUs that are healthy to retain a chain of trust.

The following Figure 4.6, shows the basics behind the above-explained protocol divided into two steps.

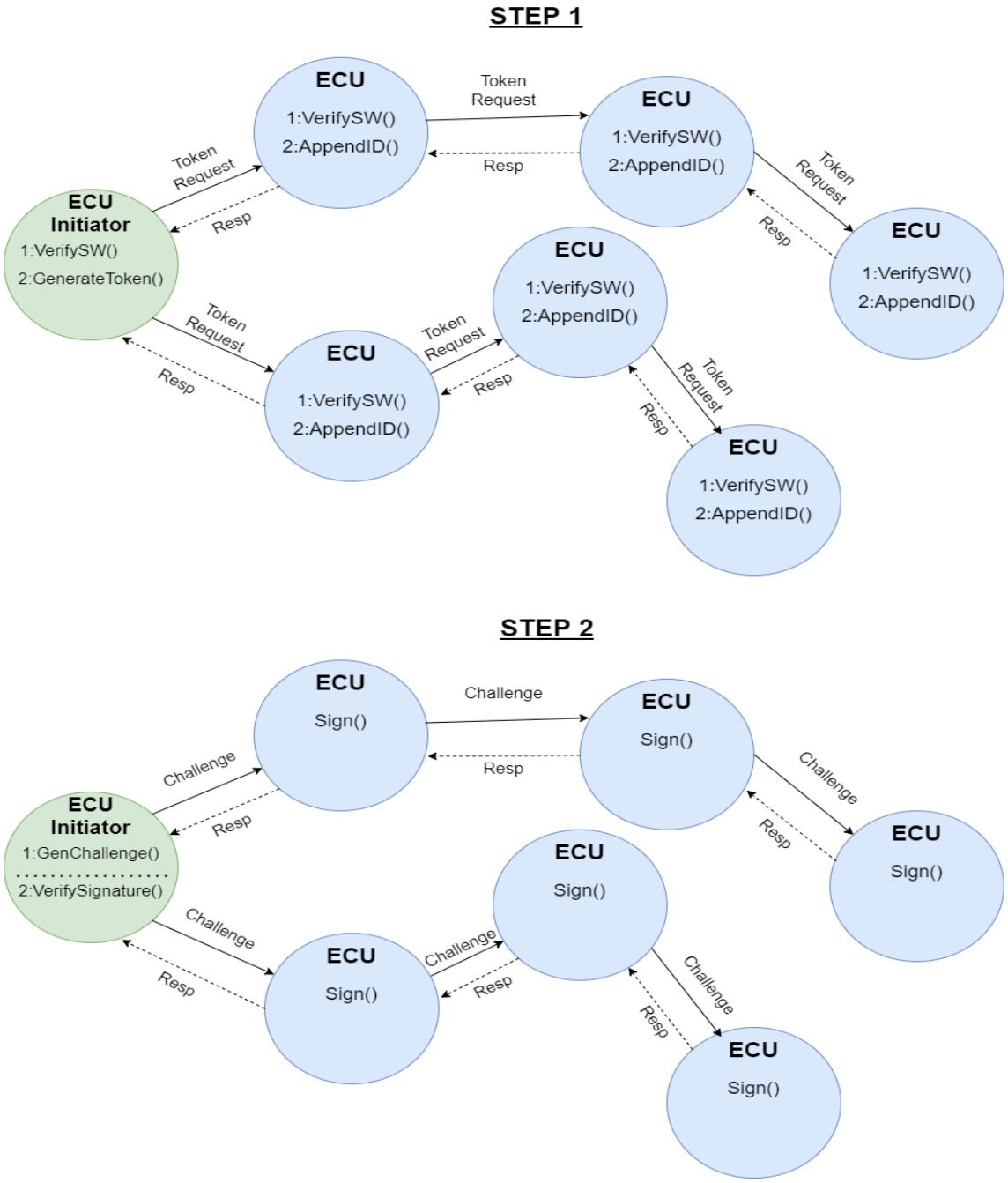

Figure 4.6: Decentralized Attestation with spanning tree and Schnorr

Another decentralized protocol that makes use of self-verification is proposed in [46]. It performs the attestation on each ECU during software updates. Specifically, the attestation is performed when the new firmware is flashed from RAM to ROM. An adversary often tries to take the chance to modify the firmware when it is stored in RAM right before it gets flashed to the ROM. But why is the firmware update stored in RAM and not ROM directly? This is because, often when downloading a system update, the vehicle is in use by the driver and that's not a good time to flash a new firmware to the ROM. The flashing should be done in a safe environment,

preferably when the vehicle is standing still. During this time when the firmware is stored in RAM, an adversary can modify it and that's why attestation should be performed directly after the flashing procedure of a new firmware according to [46].

The protocol is simple and works as follows:

1. The manufacturer generates a random challenge. It then creates a verification code by hashing the new firmware together with the challenge.
2. The manufacturer sends out the new firmware together with the verification code and the challenge to the appropriate ECU.
3. The ECU then stores the verification code and the challenge in secure read-only storage, this is possible since they are significantly smaller than the actual firmware. The firmware though needs to be stored in RAM because of its size.
4. When the ECU in a later stage wants to flash the new firmware from RAM to ROM, it follows that up by calculating the hash of the newly flashed ROM together with the stored challenge.
5. The hash is compared with the verification code and the manufacturer is notified if the firmware was modified or not.

4.3 External attestation approach

One approach to perform device attestation is to make use of the Vehicle-to-Infrastructure (V2I) model when the attestation and validation of the vehicle's devices are carried out outside of the vehicle. [3] proposes a V2I attestation protocol that requires the vehicles to be equipped with Onboard Units (OBUs) which communicates via a wireless network with Roadside Units (RSU) installed along the motorways and the attestation is performed at the edge servers. Many details will be omitted here, so for a more detailed description the reader may refer to the main article. But the concept and idea are explained below.

There are a number of stages in the protocol: Initialization phase, Registration phase, Vehicle-RSU authentication phase and Attestation phase. In the first phase, two secret numbers for the vehicle and the RSU are created by a so-called Trusted Authority to be used for the next phase. At the Registration phase, the secret numbers are used to create a pseudo-random ID of the RSU and the vehicle, as well as public and private keys for each of them. This creates the basis for the Authentication phase when both the RSU and the vehicle mutually authenticate themselves in a challenge-response manner. Finally, at the attestation phase, the vehicle's firmware is verified by using a master ECU, whose firmware, by using a checksum and a timestamp, is sent securely via the RSU to the Edge Server which in turn performs the attestation. The master ECU in this case, also needs to attest the other ECUs inside the vehicle before itself being attested. After this step is performed, the master ECU shares the result with the server which will verify the validity.

4.4 Hardware attestation approach

By hardware-based attestation it is meant the usage of special hardware on the ECUs that would allow performing attestation in a secure manner by having sensitive and secret data (passwords/keys/certificates) separated from other data and keeping it protected. Trusted Platform Module (TPM) is a widely used standard developed by Trusted Computing Group, which provides such service on computers. Basically, TPM is a chip attached to a motherboard that securely stores keys as well as generates and limits their usage [27]. Furthermore, TPM has a random number generator, creates cryptographic hashes as well as guarantees non-repudiation of the encryption operation which took place at the TPM by providing the respective certificate. Solutions proposed in [30], [38], [51], [67] and [37] were based on the usage of a TPM.

Modern CPUs are also enabling special hardware components to provide security of operations. Examples of such components are ARM's TrustedZone [62] and Intel's Intel Software Guard Extensions (SGX) [17], which put sensitive data in special regions of memory that are unreadable and separated from the rest of the memory. This allows to keep all the sensitive and secret data secure and protected, which in turn provides confidentiality, and the separation of memory on secure and non-secure parts allows the sensitive data to not be mixed with other data.

The usage of such hardware can be applicable in any attestation approach such as Master, Decentralized attestation or External attestation. The hardware helps to increase security by storing cryptographic keys at a secure and unreadable place that are used in the attestation procedure, ensuring confidentiality and integrity of the messages shared between the verifier and the prover. Usually, in the master approach, the special secure hardware is used at the master ECU, meanwhile in the decentralized protocols such hardware is used on those ECUs which need to attest themself or others. However, the cost of such hardware is usually high. The price of one TPM 2.0 in 2021 was between 15-60 US$[21].

4.5 Software attestation approach

In Section 2.3.2, the concept of software attestation was explained in a brief way. This section goes in depth on some of the most popular software attestation methods. There exists a number of software attestation methods such as, SWATT [54] , VIPER [44], [12], SAKE [55], SMATT [49], SBAP [43] and SCUBA [56].

4.5.1 Various software solutions

One of the first software attestation methods is, SWATT [54] from 2004. This protocol ensures that the only way an adversary can circumvent and hide malicious code on a device is by changing the hardware of the device. How SWATT works is that it uses a pseudorandom number generator to randomly traverse the memory based on a random challenge and add it to a checksum. This makes sure that an adversary can't predict what memory regions that will be attested beforehand.

It is also important that the memory traversal touches upon each memory region at least once, else an adversary could get away with changing a single bit in the memory. The exact number of traversed regions should be $O(n\ ln(n))$ where n is the size of the memory. Further, as explained in Section 2.3.2, software attestation measures deviations in run time, thus the verification code needs to be optimized in a way such that an adversary can't find a way of executing it any faster. This is where software attestation methods often get criticized since it relies on this vague security statement. Another difficulty is to reduce the large amount of jitter that occurs in the communication between the verifier and the prover. Jitter can lower the accuracy of the attestation method. One way of minimizing the impact of jitter is by running the algorithm iteratively in a large number of iterations.

What an adversary could try to do is to switch out the verification code against a malicious piece of code, as seen in Figure 4.7. The adversary would still need to keep the old verification code so that during attestation it can redirect the attestation of the malicious code to attest the old code instead. According to the paper [54] this will force the adversary to insert extra if statements for checking which memory region that should be attested. And, that will be detected as an increase in run time thus the attack fails.

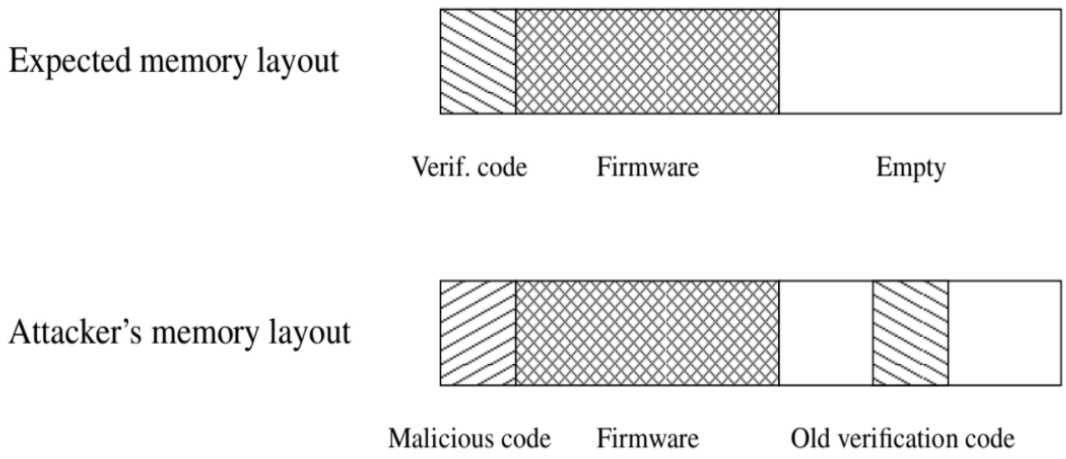

Figure 4.7: Example of attack on SWATT [54]

A drawback of this approach is that the run time increases linearly with the number of memory accesses. This could lead to high time consumption for large memory areas. Another disadvantage is that the verifier needs to perform the same procedure as the prover thus also having a copy of the prover's legitimate memory. In SMATT [49] the authors proposed a solution to the computational overhead that could arise at the verifier during the verification process. Instead of letting the verifier do the same computations as the prover, they suggest that the verifier compares the response from the prover to responses from other provers. By applying this method the verifier could classify a response from a prover as faulty if it deviates from the

majority of responses. This approach assumes that the same challenge is sent to all provers.

Xinyu Yang et al. [69] have proposed another software attestation method, called low-cost remote attestation scheme (LRMA) for smart grids (also known as the electricity network) which is based on SWATT. However, in comparison to SWATT, the main change is done in the challenge-response stage of the protocol in a way that LRMA is more applicable for smart grids, it takes into account time delays over the network and adjusts the attestation rate of the devices to the probability of compromising of device. Specifically, the verifier collects attestation results from each node in the network over a time period, and for each node it computes attestation failure probability as well as assigns a risk scale to each device (the more failed attestation on the node, the higher risk and failure probability the node is assigned). Based on this scale, the verifier adjusts its attestation rate for the future. Thus, during the next attestation processes, the nodes with higher risks will be attested more often, meanwhile frequency of attesting "low-risk" nodes will be accordingly lower. Such an approach, according to the authors, can reduce the total number of attestations and will allow to focus more on "risky" nodes rather than on "safe" ones. The attestation by itself in LRMA is performed via challenge-response: verifier sends a challenge, starts to measure the time; prover receives a challenge, computes the checksum of its firmware, sends it back, to the verifier; the verifier checks the checksum for the correctness and that the time is within the time frame. If the attestation is successful, the risk scale on such node will be decreased, otherwise the risk scale will be increased. For a detailed description of the protocol we would refer the reader to the main article, here we just presented the basic concept. It is worth mentioning again that LRMA is done for smart-grid networks, and this type of network looks a bit different in comparison to the CAN bus. For example, for a message going from one node to another sometimes it will be needed to pass other nodes in a smart-grid network, meanwhile in the CAN bus each node is connected to one single bus.

One approach to software-based attestation was described in the paper [44] and is called VIPER . This protocol was constructed for the integrity verification of peripheral devices' firmware in the computer and is based on the time and latency of the transmitted messages between the verifier (CPU) and the prover (peripheral device). The purpose of VIPER is to eliminate proxy attacks, which is when the prover uses the computational power of another device to circumvent attestation failure. Figure 4.8 demonstrates the attestation environment. The verifier has a nonce-generator, the respective checksum function and the copies of the devices' firmware hashes. The attestation is started by the verifier which sends a special request to the prover with the nonce (challenge) and starts the timer at the same time. The prover computes the checksum of the nonce and sends the result to the verifier. This can be considered a challenge-response procedure when the verifier makes sure that there is an untampered execution environment in the device to be attested. After the prover sends the checksum to the verifier, it also starts to compute the hash of its firmware. Upon the arrival of the checksum, the verifier checks

if the checksum is correct and the time frame is not exceeded. When the verifier receives the hash of the prover's firmware it verifies this as well.

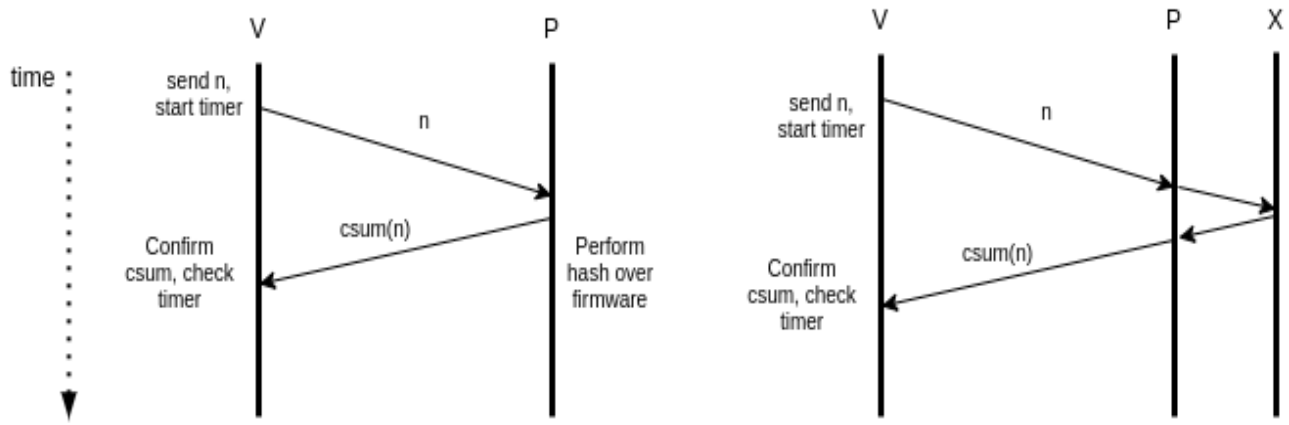

Figure 4.8: Challenge-response in normal (left) and proxy attack (right) scenario. V is verifier, P is prover, X is proxy, n is nonce (challenge), csum is checksum

The risk in the above-mentioned environment is that a tampered device can use a "proxy server" with unlimited power which can make all the computations (see Figure 4.8 right part). To eliminate this risk VIPER suggests sending an additional nonce (challenge) before the arrival of the checksum in order to increase the communication delay (see Figure 4.9). The checksum function is an array of bit vectors to have an efficient message transfer. The prover sends only a limited number of vectors which are selected randomly and, according to VIPER, the last checksum vector to be sent depends on the second challenge received. Figure 4.9 demonstrates how this works. According to the authors such an approach will force the proxy device to return all the checksums of the correct vectors in order to not miss the deadline, which in turn will lead to time delay and detection of the proxy attack.

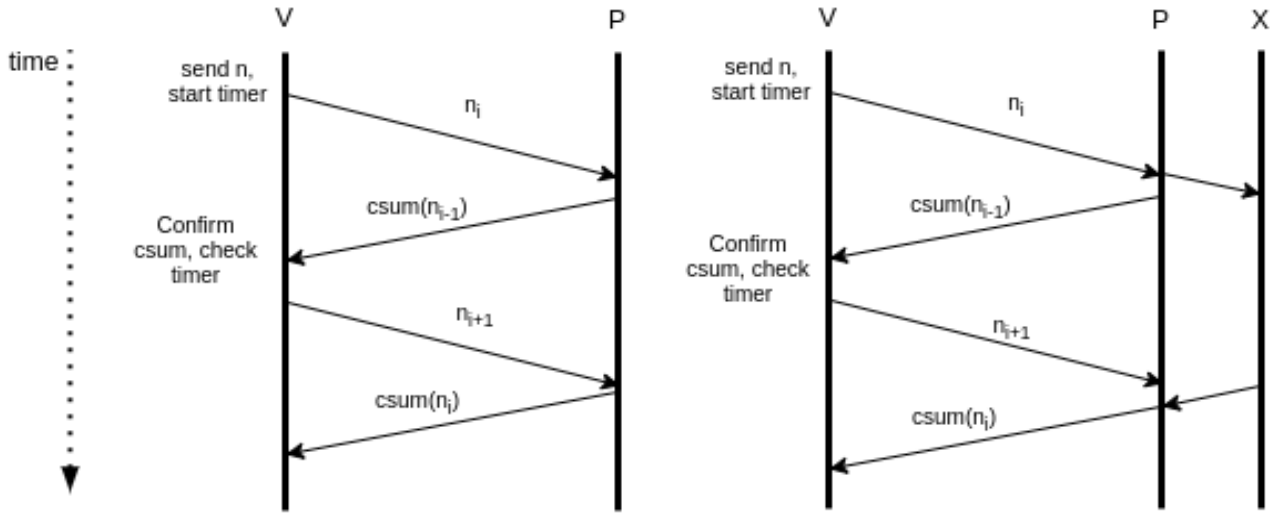

Figure 4.9: Challenge-response with VIPER in normal (left) and proxy attack (right) scenario. V is verifier, P is prover, X is proxy, n is nonce (challenge), csum is checksum

Two attestation protocols called SAKE [55] and SCUBA [56] uses a method called indisputable code execution (ICE). The method aims at providing the verifier with a guarantee that the code running at the prover is unmodified by any adversary. ICE is what's called a self-checksumming code which uses a similar approach as SWATT [54] but instead of creating a checksum of the whole memory it creates one of itself. In combination with ICE they also use the Guy Fawkes protocol [10] to achieve mutual authentication between the verifier and the prover. This is a protocol that uses a hash chain method to achieve mutual authentication.

SBAP [43] basically works in a similar way as SWATT [54] but the difference is that it fills the data memory with pseudo-random bits generated by the random challenge. And, in the next step, it attests both the program and the data memory to make sure there isn't any malicious code present.
Claude Castelluccia et al. [16] have shown that there exist various vulnerabilities with software-based attestation. They also showed some methods that can be used to hide malicious code without being detected by the attestation method. One of these methods uses return-oriented programming to move the malicious code from non-executable memory to program memory. Another proposed method is to compress the program memory to be able to fit malicious code.

4.5.2 Elimination of physical attacks

To perform physical attacks on a device requires an adversary to have physical access to that device and by having that the adversary can tamper with its hardware. Generally, software attestation approaches are less resilient against these types of attacks. In order to eliminate these attacks a protocol SCAPI in [33] was proposed.

When a device is physically tampered by an adversary the device is often taken offline. All devices have a common session key which is constantly updated by one leader device. It is assumed that each device has a pair of session keys: the current session key (sk_{cur}) and the next session key (sk_{next}). The leader device periodically authenticates each device by looking at whether the device's (sk_{cur}) is valid (i.e. the latest one) and, in case it is, the leader generates a new session key and sends it to the device. When a device receives a new session key from the leader, it updates sk_{cur} with its sk_{next}, and sk_{next} gets a value of the newly arrived session key from the leader. In case the leader identified the device's sk_{cur} to be not valid, this means that the device was offline for some period of time and was not able to receive the latest key update, therefore this device was physically compromised.

Wenwen Yan at al. [68] proposed a similar protocol, EAPA, it works in a decentralized way where all devices constantly send out heartbeats to their neighbors. This is in contrast to SCAPI which worked in a centralized way, thus exposed to a single point of failure. In EAPA all devices act as both provers and verifiers. At first, each device acts as a prover and sends a heartbeat to all its neighbors. The devices then act as verifiers and verify all received heartbeats. If the heartbeat is insufficient or lacking, that device is reported to the central authority as compromised. The authority then makes the decision to remove the compromised device from the network.

4.6 Hybrid attestation approach

Hybrid attestation is a form of attestation where pure software based methods which was explained in Section 4.5 is strengthen by combining it with lightweight hardware. Examples of these methods are PUFAtt[35], [53], ATT-Auth[4] and AAoT[22].

These methods combine the hardware feature of PUF with pure software-based attestation methods such as SWATT which is explained in Section 4.5. PUF was explained in Section 2.3.2 and software-based solutions were explained in Section 4.5. One of the main problems with pure software-based solutions is the strong assumptions that it comes with. One assumption is that a prover cannot be impersonated or collude with a stronger device. A more powerful device could be able to execute a malicious piece of code in the correct time frame and thus be able to circumvent the attestation protocol. To prevent this, the hybrid attestation methods make use of a PUF which is unclonable and cannot be impersonated by another hardware device. This adds authentication to the attestation protocol and prevents physical attacks.

What PUFAtt[35], ATT-Auth[4] and [53] does is that they include the PUF in each iteration of the memory checksum calculation. This is done to tie each iteration to the specific hardware of the prover device. Specifically, the response from the PUF is used as input into for example SWATT to generate the memory traversal path. This makes it infeasible to generate the correct path on another hardware device.

In AAoT [22] , the author explains that the setup phase of the previously explained

methods would be computationally heavy since the PUF is used in each iteration of the checksum calculation, thus a large amount of CRPs need to be shared between the verifier and the prover. AAoT brings a solution to this by only using two CRPs instead of 82,020 which is the case for the other methods. To compute thousands of CRPs per attestation would be nearly infeasible on low-end embedded devices which are common in vehicles. Another improvement in AAoT is the mutual authentication between the verifier and the prover. This can help to mitigate malicious verifiers.

There also exists other Hybrid attestation methods such as HYDRA [19]. Since HYDRA builds on top of already verified and secure hardware by leveraging the L4 Micro-kernel it is not suitable for use in low-end devices. De Oliveira Nunes et al. in [47] proposed a number of properties that need to be held in order to have a secure hybrid attestation. One of these is key protection which implies:

- **Access Control** meaning that the secret key must be accessible only to the attestation software.
- **No leakage** implies the storage of the key only in the secure memory after the key was accessed by the attestation software.
- **Secure reset** indicates the erasure of the memory registers which were affected by the prover's secret key if there was a reset during the attestation process.

Another property defined by the authors was safe execution which has the following criteria:

- **Functional correctness**. The attestation protocol will guarantee that the attestation code on the prover will run as it is expected to run.
- **Immutability**. The attestation code must be immutable.
- **Atomicity**. The execution of the attestation code will now be interrupted.
- **Control invocation**. The attestation code will start from the first instruction and will end on the last instruction.

An example of a protocol that fulfills such properties is the dynamic protocol SMART [20] which is described in Section 4.7.

In several proposed methods, the attestation process consists of a step where the firmware of the attesting device is hashed and then sent to a verifier. To calculate a hash over a large memory section can become a computationally intensive operation [5]. Because of this, another method for verifying large software could be needed. This is especially applicable for low-end embedded systems where there are relatively restricted resources.

One method to solve this issue is by using a random sampling approach. The hybrid attestation method, HAtt, that was proposed by [5] makes use of a challenge that is used to choose a random number of sampling bits from the memory. The drawback with this approach is that the verifier also needs to be in possession of the entire firmware to be able to perform the same sampling in order to verify the validity of the sample. But, by using this approach, some heavy computations can be moved from the prover to either a stronger master ECU or a center which is in possession of more resources.

The HAtt protocol works as follows:

1. It assumes that the Verifier (V) is in possession of challenge-response pairs (CRP) that is calculated in the setup phase with the provers (P) PUF and stored at V.
2. Firstly V sends a challenge and two random nonces to P.
3. P then generates a response R by running the challenge through the PUF. P then generates the sequence of blocks that should be attested by combining R and one of the random nonces.
4. A number of bits in each block is then picked based on the second random nonce. These bits are added to a bitstring (B).
5. B is then sent from P to V
6. V then needs to go through the same process to be able to validate B, but instead of using the PUF, it fetches the response from the CRP which is stored in V.

4.7 Dynamic attestation approach

There are two ways of attesting a device, either static or dynamic. The methods which are described above are considered to be static and this is the most common attestation method. Static attestation implicates that the attestation is performed on the initially loaded software on the device before it is executed. By doing this the verifier can make sure that the software hasn't been altered before running it. This prevents malicious code injected by an adversary from running on the device.

What static attestation can't detect are run-time attacks which can be performed if an adversary gains control over the stack or the heap. What dynamic attestation does to prevent the just mentioned attack is to monitor the control flow and isolation of the software to verify a correct and untampered execution. A known problem with dynamic attestation is that it often comes with significant overhead for both the prover and the verifier [2]. This is because the execution path needs to be stored during the execution of the software. Some researchers have tried to solve this issue by using a cumulative hash of the execution path to minimize the computational overhead. Examples of such methods are, DIAT[2], LO-FAT[18] and C-FLAT[1]. At the verifier though, the task of searching the control-flow graph (CFG) which stores all possible execution paths of the prover, is still complex and needs computational resources. What these methods try to do is to move the heavy computations from the low-resource devices to the high-resource devices.

A method SMART , Secure and Minimal Architecture for (Establishing a Dynamic) Root of Trust, for dynamic remote attestation with efficiency, simplicity and security in mind was proposed in the article [20]. The idea behind this protocol was to create an attestation method based on the interaction between hardware and software, which would be applicable for low-end embedded devices without memory management and will guarantee untampered execution of the attestation code on the prover's device. SMART assumes that there is a shared secret key, K, between prover and verifier. K is stored in the Read-Only memory on the prover's device

and can be accessed only from there and only by SMART code in ROM. In case of violation of this assumption, the device will perform a hardware reset.

Figure 4.10 demonstrates how the SMART works. The protocol is initiated by the verifier, V, which sends a special message to the prover, P, consisting of 1) memory boundaries of the attestation (a, b), 2) memory address to where the code execution will go after attestation is performed (x) if x_{flag} is set, and 3) nonce (n) for avoiding replay-attacks. Later on, P activates its attestation program in ROM and computes checksum (C) on the memory region $[a, b]$ with n. The control flow goes to x and C is sent to V, which in turn verifies C. It is worth mentioning that when x_{flag} is set all the interruptions are disabled when the execution goes outside of the ROM which prevents TOCTTOU attacks. The authors point especially to the importance of having isolated K which can be accessed only when the code execution is inside the ROM. In order to achieve this, they propose to enable connection from the data bus to K only when the program counter (PC) is pointing to the address within the ROM range and the data pointer is pointing to the address range of K. Also, in order to prevent ROP attacks, the authors proposed an additional feature that implicates that PC is allowed to enter ROM only at the address where SMART starts and PC may leave ROM only at the address where SMART ends. This, according to the authors, can create a secure control-flow environment.

The Figure 4.10 demonstrates and visualises how the SMART protocol works:

1. V initiates attestation and sends the respective parameters;
2. P starts SMART execution;
3. P performs checksum over memory within [a,b] range and by using K stored in protected memory, the program counter (PC) must be in ROM in order to access the key;
4. Attestation result C is stored;
5. P sends the result to V which in turn verifies it;

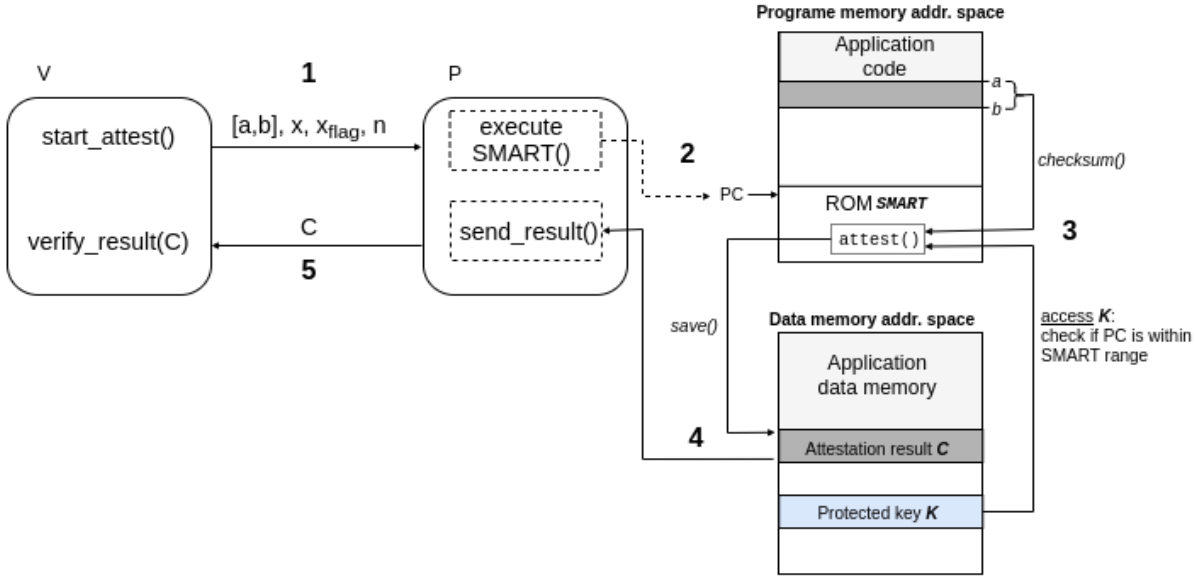

Figure 4.10: SMART protocol. V is verifier, P is prover.

5

Proposed attestation methods

This chapter presents the proposed attestation methods which are based on the performed research and analysis of the results described in Section 4. Further, this chapter includes new features and perspectives which were gained during interviews with industry experts.

5.1 When to perform attestation

The attestation needs to constantly be performed in order to keep the vehicle secure. Different kinds of ECUs can be attested in different situations, both to save time and computational resources. Attestation cannot be performed on those ECUs which are not idle and are executing some operation. Performing attestation, in this case, would block the vehicle's capability to do its basic functions. Therefore, we need to define situations when the attestation of ECUs can be performed. In the following situations the attestation can be performed:

- Vehicle's start up/unlock
- Driving
- Software updates

During the first case, when the vehicle is started up or unlocked, the most critical ECUs need to be attested right away in order to get the vehicle ready for driving as soon as possible. By the critical ECUs we mean those ECUs that are needed to drive the vehicle in a safe manner (examples: braking, acceleration, turning). Such prioritization improves the attestation speed which is highly important especially in the start-up phase when the driver wants to use the vehicle immediately. Later on, after critical ECUs are attested, other ECUs with non-critical functionality can be attested. Non-critical ECUs can be attested either during the vehicle's startup (in case the critical ECUs were attested within a few seconds), or during the movement when ECUs are not performing any task.

ECUs can also be attested periodically during the driving when they are idle in order to certify that no hacking was performed during the movement. In this case, the attestation should be very quick in order to not occupy the device's resources for too long when they are needed. The last scenario when the ECUs need to be attested is after software updates. Software updates are often flashed when the vehicle is in an idle state which also is a good time to perform the attestation.

5.2 Attestation protocols

This section explains two device attestation protocols that we propose for the vehicle industry. Both protocols are similar to each other, but have some conceptual differences. Also, both protocols are based on various techniques described in Section 4. For the sake of simplicity and abstraction, we call these protocols as **Protocol 1** and **Protocol 2**.

5.2.1 Protocol 1

Protocol 1 is a simple centralized attestation protocol. In Figure 5.1 the architecture is shown and below is an overview of how the protocol works:

- One powerful Master ECU uses either a Hybrid protocol, such as AAoT, or a Software protocol, such as VIPER, to internally attest the other ECUs. Which method depends on the resources at each ECU.
- Hybrid attestation should be performed on ECUs where a SRAM chip is available. ECUs without SRAM are attested with a software type of attestation.
- An external party can then remotely attest solely the Master ECU which has more hardware security.
- During the external attestation the Master ECU sends a report on how the internal attestation went.
- Symmetric encryption is used in the communication between the Master ECU and the external party since it is significantly faster than asymmetric encryption. A PUF is used to create the symmetric keys.
- Only the Master ECU stores the current firmware of each ECU in ROM for the verification phase.
- This protocol is a combination of both hardware, software and hybrid attestation, but limits the secure and costly hardware to only one powerful ECU. It also includes External attestation and Master ECU attestation.

The main goal of this protocol is to make as small modifications to the existing hardware as possible at the same time as being highly efficient and secure. Protocol 1 should be able to be applied inside already driving vehicles for a low cost.

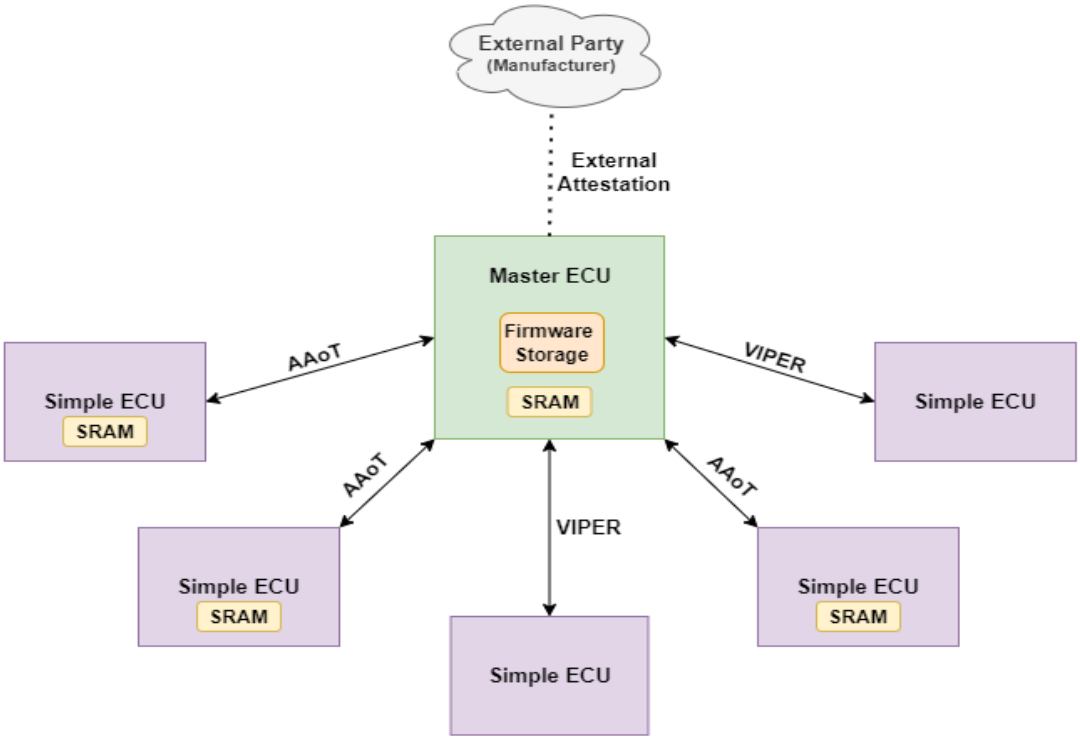

Figure 5.1: Protocol 1 architecture. ECUs with SRAM are attested with AAoT and others are attested with VIPER

5.2.2 Protocol 2

The following assumptions are made for the Protocol 2:

- There is a single central powerful ECU, *master*, which has a random generator function and stores information about each device in the vehicle (ID, attestation key, valid firmware version, SRAM of each ECU). Master is also able to communicate with a remote server and receive updates.
- There are around 4-8 of special ECUs, *half-masters*, connected to the master. Each half-master is responsible for a small area, *cluster*, which consists of a number of simple ECUs. Half-master serves as a bridge between ECUs in the cluster and master ECU. The specified number of half-masters derives from the future in-vehicle architecture where there is a limited number of gateways (around 4) which will be placed in different parts of the vehicle and connect multiple ECUs (see Section 2.1). Half-masters have a random generator function, unique attestation key AK (stored in SRAM PUF which, in turn, creates protected memory) and boot nonce N_b. AK is created by SRAM PUF when the master sends a nonce to the half-master.
- Each simple ECU inside of a cluster is either *critical* or *non-critical*, depending on the functionality and its impotence to the vehicle's safety. The critical ECUs are those devices which have the highest importance for the vehicle's movement capability and make the driver safe. Examples of such ECUs are

braking, acceleration, engine, etc. According to the protocol, such ECUs are also equipped with SRAM. Other ECUs are considered as non-critical.

- Each simple ECU has a risk scale, RS, which depends on two factors:
  1) criticality c, how critical ECU is for the vehicle's safety ($1 \leqslant c \leqslant 5$): *none-low-medium-high-very high*, where 1 is *none* and 5 is *very high*. All *critical* ECUs fall under category *high* and *very high*. This scale was chosen based on the proposal from ISO26262 standard.
  2) failure rate fr: how many times the ECU has been successfully verified or failed ($1 \leqslant fr \leqslant 10$).

  In other words, $RS = c + fr$. This scale is later used to determine the frequency of the attestation. The criticality range was chosen between 1 and 3 because of the fact that an ECU can either have a direct impact on vehicular safety (braking) or non-impact at all (air conditioning); at the same time, there can be an ECU which has a non-direct impact on safety, but can affect safety in some way (example: windshield washer on the front window which can worsen driver's visibility and affecting the safety). Therefore three types of criticality were chosen. As for the failure rate, the scale between 1 and 10 is a simple scale choice, without being too distributed, to show clearly the difference in how often the ECUs can fail attestation.

The protocol is divided into three phases.

Phase 1.
The first phase of the protocol is divided into three steps, as shown in the Figure 5.2:

- Attestation is initialized by the master which sends a challenge to half-masters as described in ALE protocol (step 1 in the figure). Together with the challenge, the master sends memory bounders on which attestation needs to be performed (see method SMART) and all other parameters that SMART assumes.
- Half-master, upon the receiving of the challenge, creates a new attestation key AK with PUF. This is a bit different from ALE protocol where prover has constant AK, meanwhile in Protocol 2 AK changes on each session. Then, half-master calculates its hash as described in ALE. The attestation key is accessed in a way as described in SMART. When the hash is calculated, half-master sends a response to the master which performs the verification (step 2).
- If the verification of half-masters' firmware was successful, the master sends a list of valid firmware hashes of the simple ECUs within half-master's cluster (step 3).

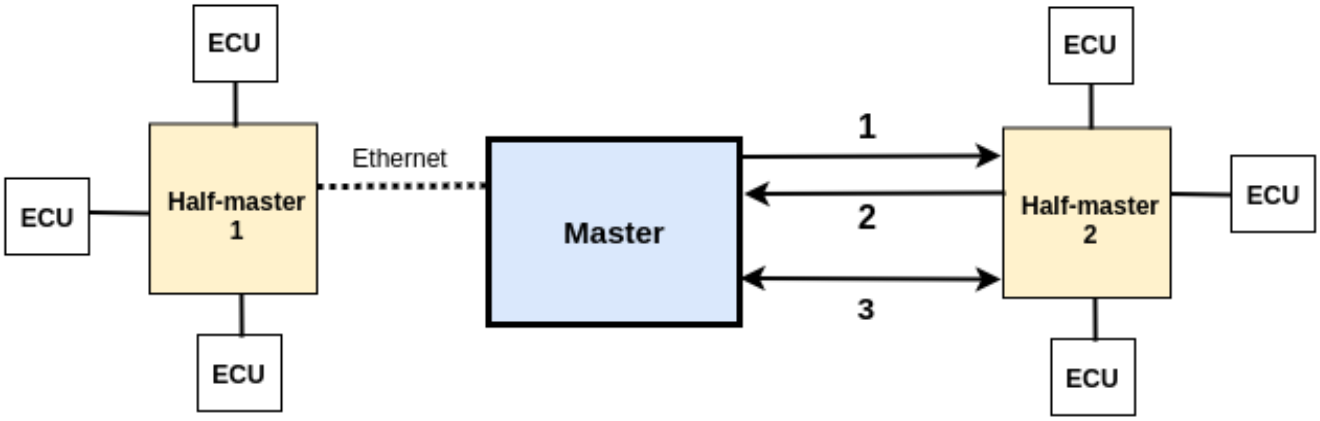

Figure 5.2: Protocol 2 - Phase 1

Phase 2:
When the firmware on the half-masters has been verified, the attestation is performed on the simple ECUs. This is the second phase of the protocol, which is also divided into two steps and shown in Figure 5.3.

- Each half-master attests simple ECUs in its cluster. There are two types of simple ECUs: critical ECUs (shown in red color in the figure) and non-critical. Critical ECUs are attested with AAoT and non-critical are attested with VIPER. The priority in attestation order is given to critical ECUs by sending attestation requests first to the critical ECUs.

- If the ECU was verified, the half-master sends a grant to the ECU as permission to operate. Also, depending on the verification results of the ECUs, the half-master ECU assigns a risk grade to each simple ECU by either incrementing or decrementing fr after failed or successful attestation respectively, but fr can never be below 1 and above 10. If the attestation failed, the attestation attempt should be repeated one more time.

- Half-master collects the results of the attestation of the simple ECUs and sends them to the master in the form of a report/scheme.

Figure 5.3 visualises how **Phase 2** of the Protocol 2 works. The priority is given to the critical ECU which is shown in the figure in red color. This ECU is attested with the AAoT protocol (step 1 in the figure). Then, Half-master attests other ECUs with VIPER (step 2). Finally, the attestation results are sent to the master (step 3).

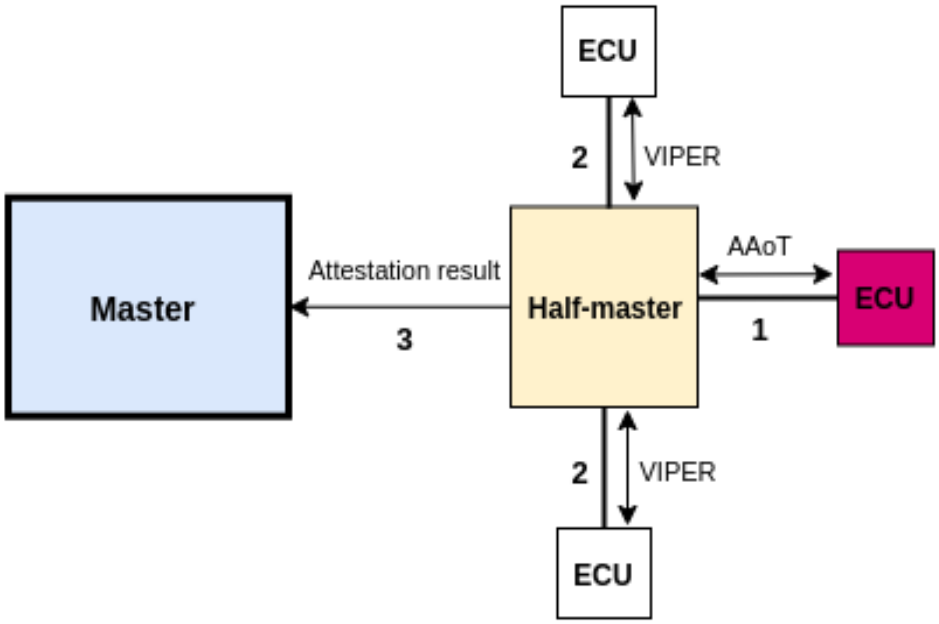

Figure 5.3: Protocol 2 - Second phase. ECU with red color is a critical ECU.

Phase 3:
When attestation is performed on all ECUs, the final phase includes a periodical attestation of some of the devices, checking of their vitality as well as getting updates from a remote server. This phase is shown in Figure 5.4:

- Master periodically sends a heartbeat to half-masters, and half-master send periodically heartbeat to the critical ECUs (the heartbeat is shown with a heart symbol in the figure). This is done to prevent physical attacks which can be read more about in Section 4.5.2
- ECUs with a high risk scale ($RS >= 5$) are attested more frequently, even during the movement of the vehicle. Such ECUs are shown in the red color in the figure. The reason for attesting more frequently and during the movement of the vehicle is since we have seen proof of remote attacks such as the Jeep hack [45] explained in Section 1.1. We define the high risk scale to be 5 or more, because in this case we can guarantee that all critical ECUs will be attested during this phase of the protocol, as well as other ECUs which failed attestation several times.
- Master only communicates with the remote server in order to get updates or when there has been an internal attestation failure. When the manufacturer releases an update of the firmware, the firmware is sent together with a challenge and $hash(new\_firmware + challenge)$ and the new firmware is flashed to the ROM as described in [46]. Also, during each update, the remote server sends an updated list of valid software versions for each ECU to the master.

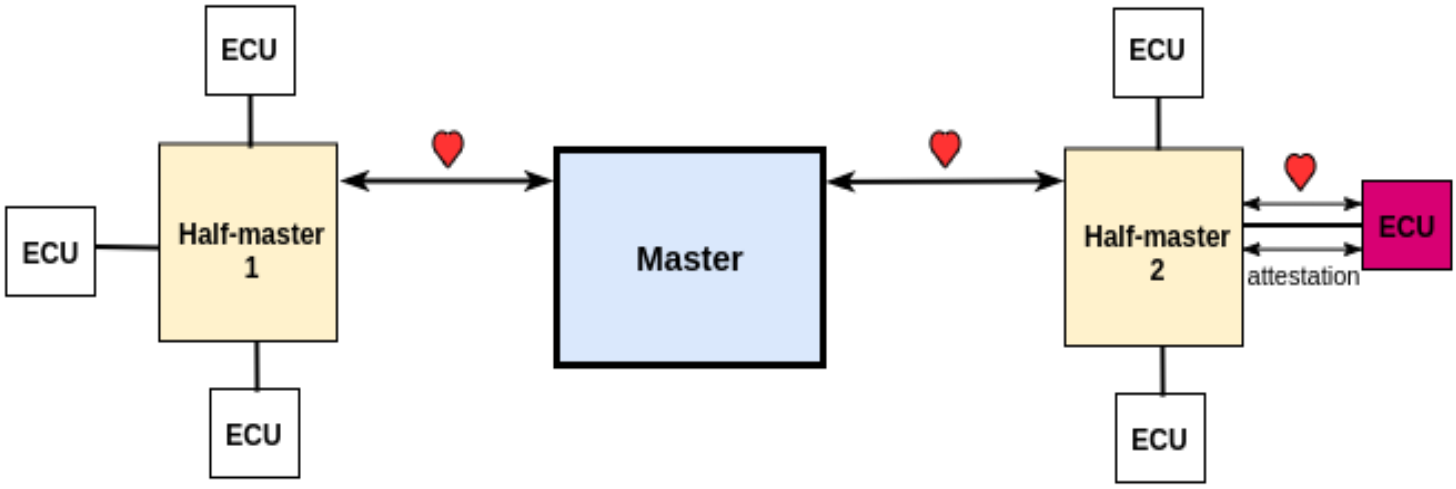

Figure 5.4: Protocol 2 - Phase 3

5.3 Feasibility

This section presents feedback that was gained during interviews with industry experts in the cybersecurity area. It mostly revolves around the feasibility of implementing the proposed protocols and what problems that potentially could arise.

One potential risk of using VIPER and AAoT is that it could be hard to get a stable response time between the prover and the verifier. This can for example be because of the prioritization approach that is used on CAN messages. A message with higher priority could force a VIPER/AAoT response to wait thus creating jitter/delay in the communication between the prover and the verifier. This can also be highly unpredictable and hard to solve. One way of solving this could be to increase the time frame in which we allow response messages, but this can allow compromised provers to slip through occasionally. Another solution would be to assign a higher priority to the attestation messages on the CAN bus which would lead to fewer delays, but could result in delays for other critical messages. On the other hand, if the attestation is performed on the vehicle's start-up, there should be no high-priority messages except the attestation messages.

Another reflection that was made by the experts was that the Master and Half-Masters need to have complete knowledge of the firmware of each ECU that they attest. Furthermore, in VIPER and AAoT both the verifier and the prover perform the same computations (computing a hash of the prover's firmware), since the verifier needs to recalculate the valid hash in order to verify the received hash. Both previous statements are true which means that the verifiers will require some increased storage capacity and computational power, but the trend in the vehicle industry is going towards more powerful ECUs.

A general feedback about the Heartbeat, which is used to prevent physical attacks, is that there could possibly arise situations of False Positives. For example, an ECU could possibly go offline for other reasons than physical tampering. It could be that the ECU needs to reboot for some unknown reason and thus is unable to send a heartbeat in time. This must be taken into account if a solution like this is to be implemented.

Another thing that was pointed out was the use of PUF/SRAM inside critical ECUs. Adding extra components like SRAM can be hard to get approved from a cost perspective, but there are currently plans of integrating Hardware Security Modules (HSM) inside critical ECUs in the future. HSM could then potentially be used instead of SRAM.

Overall, the feedback on the protocols was positive and according to the experts, the protocols are feasible to implement if the above-mentioned feedback is taken into account. A specific compliment to the protocols was the advantage of not requiring major modifications to the existing hardware.

6
Discussion

This chapter presents a discussion over achieved results in the project and proposed protocols as well as the applicability of the various attestation methods to the vehicular industry. Also, different aspects of device attestation such as state of the art, latest trends and alternative solutions are discussed in this chapter.

6.1 Attestation methods and their applicability

In this section, the attestation categories from the Literature review results in Section 4 are elaborated on and discussed. Further, it mentions pros and cons with the previously described attestation approaches.

6.1.1 Master ECU approach

The attestation category Master ECU Approach, described in Section 4.1, is a common way of performing device attestation. An advantage with this approach is that it limits the amount of resources needed since it only requires one powerful ECU. Because of this, it could be a relatively cost-efficient approach if applied correctly. A disadvantage is the single point of failure and that the master ECU will be an attractive target for adversaries, since it is the core of the attestation. One of the schemes that uses the master ECU approach is ALE. This protocol requires, enabling some properties such as immutable bootloader, exclusive key access for bootloader and interruptibility, which together fulfill requirements of Exclusive Access to Secret key, Uninterruptability and Immutability, described in the Section 2.4.1. These properties make ALE applicable for lightweight devices and do not need special expensive hardware, which results in a low cost. Since ALE makes use of its unique attestation key which is stored securely, prover creates a response key that cannot be forged by anyone else; this also fulfills unforgeability and no-leak requirement. ALE devides ECUs in two categories: safety-critical and non-safety-critical. Such a division can be useful because it allows to prioritize the available resources to primarily attest the most important ECUs. Another advantage of ALE is that it makes sure that a new nonce and response key are generated for each session which makes the scheme resistant to replay attacks.

Another master ECU approach was used in [48] which establishes a remote connection to an external party (remote server). In this approach, the remote server delivers all information needed to perform the attestation, therefore the computa-

tional power and memory needed on the master ECU can be reduced and can be held in the cloud. Furthermore, this protocol allows to establish a symmetric key between any pair of ECUs inside the vehicle, which is more efficient than an asymmetric approach, since symmetric encryption operations are quicker. In addition to this, the cloud delivers new keys and new key matrices for each session which also prevents replay attacks. Another advantage is that each message contains a secrete r_2 (see Figure 4.3), which is a proof of successful attestation by master ECU, and a counter, which prevents replay attacks.

At the same time, having a remote connection to the cloud/server can be considered risky in a case when the connection is not established in a proper way which can result in a non-secure communication channel. Furthermore, this approach fully relies on the fact that the remote server works properly, if the remote server is broken or hacked it means that the whole attestation process will fail.

6.1.2 Decentralized attestation approach

In contrast to the Master ECU Approach, there is the Decentralized Approach which works in the opposite way. Instead of putting all the load on one ECU, the load is distributed throughout multiple ECUs. This is done as a response to the single point of failure, thus making the approach more resistant to denial of service attacks. A disadvantage is that this requires more resources at each individual ECU since each and everyone of them should be able to act as a verifier of the IVN.

The protocol described in [29] is an example of a decentralized protocol where the attestation process is performed within a so-called *attestation groups*. Such an approach allows to categorize a big number of ECUs into different groups according to their functionality, which creates clusterization of co-related ECUs. Thus, the attestation is performed only on a limited number of ECUs within the clusters, which allows ECUs in such clusters to not waste energy on attesting unnecessary ECUs which they do not depend on, but instead they can focus on and put their resources on those ECUs which are needed to be attested. Furthermore, the attestation can be performed in parallel in a way that each attestation group can perform attestation independently, which increases the efficiency of the attestation process in the vehicle. Another advantage of the protocol in [29] is the constant change of keys which prevents replay attacks, usage of symmetric keys which makes communication more efficient, and usage of MAC which guarantees the unforgeability requirement.

However, a significant disadvantage of the above-mentioned protocol is that it requires special secure hardware which is costly, but at the same time such hardware makes attestation more secure. Thus, from a cost perspective this approach could be more expensive than a centralized (master ECU) approach. The authors in [29] have also mentioned that the protocol can be performed either sequentially or in parallel. Based on their experiment results (presented above in Section 4), we can conclude that the sequential approach is too slow to meet the automotive requirements where the attestation should happen within a few seconds. Thus the parallel solution is

the preferred one. As an alternative solution to [29], the attestation groups can be created based on their geometrical location instead of functionality, which will put ECUs within one attestation group close to each other and this will affect the efficiency of communication. Despite being decentralized, the protocol in [29] can be considered in some way to be using the master ECU approach, since there is one node which acts as the initiator of the attestation process and this node also acts as a verifier of other nodes. On the other hand, because each node can initiate the attestation, this protocol can be considered as decentralized.

Another distributed algorithm SALAD [33] makes use of a spanning tree to perform the attestation. The firmware verification process here can be considered as a chain reaction when one node initiates the attestation process, then other nodes in the topology gradually join the procedure and exchange their verification results. This allows having a parallel process of attestation which results in an efficient attestation in a big topology. The main obvious advantage of this decentralized protocol is that it allows for each node to see the attestation results of every other node in the network. In addition to that, the protocol is capable of mitigating physical attacks that compromise all devices, since each device is using its unique key for the attestation and an adversary can only forge the verification for those devices that it got physical access to, but not the other devices. Moreover, SALAD ensures the attestation on the device is performed only if the phase id (*pid*) of the requesting device is higher than the device's phase id. This eliminates the risk of replay attacks. On the other hand, this protocol is more suited for topologies like smart-grid networks where not all of the nodes are directly connected. In the case of vehicles, where the CAN-bus is used, this method is not appropriate, since if one node (ECU) sends a message, all nodes will receive it. Therefore, the chain-reaction of message delivery is not relevant for the CAN bus. However, the property of sharing "reports" of the collected attestation results between the nodes can be useful, since in this case each node will be able to find a Byzantine-faulty node(s) [32] just by comparing received reports from different nodes. But this can work only if each pair of ECUs has a unique and secret symmetric key for their communication. Last, but not least, SALAD also requires special hardware on all ECUs, since the key must be stored securely, which is costly, but at the same time it will fulfill the requirement for Exclusive Key Access.

The decentralized protocol PASTA works in a similar manner as SALAD. There is one initiator which sends a request (token) to neighboring nodes and the neighboring nodes send the attestation request to their neighbors. This protocol achieves high security and manages to distribute the workload onto different ECUs. It also mitigates the vulnerability of a single point of failure. However, it is dependent on a large amount of secure hardware and time-consuming cryptographic algorithms. This violates the low-cost and high-performance requirements of in-vehicle ECUs. Similar to SALAD, PASTA is not appropriate for a CAN bus, since there is no need to retransmit the attestation request by each node. However, one possible solution for that (i.e. enabling PASTA in the vehicle) could be that the attestation is performed in a way where there are several initiators which send requests to some specific ECUs. But in this case, the protocol would be similar to the one described in [29]. Additional cost with the hardware, needed for PASTA, makes this algorithm

costly to implement.

A protocol described in [46] performs attestation when the manufacturer releases a software update. In this case, a decentralized attestation process is achieved in a way that each ECU performs attestation independently when it receives the update. At the same time, such an approach requires the manufacturer to establish a connection with a specific ECU in the vehicle whose software needs to be updated. To some extent, there exist similar solutions in the industry where an ECU can verify that the update is sent by a trusted party by using a signature, but this is not always sufficient. The vulnerability that this protocol mitigates is the risk of software modification by an adversary while the updated software is stored in RAM right before it is flashed onto the ROM. By sending a challenge to the vehicle's ECU, the protocol prevents replay attacks. At the same time, the protocol can be questioned in the following aspects. Firstly, since the attestation is performed only in conjunction with software updates, it means that attestation is not performed very often unless new software releases are done often. Secondly, in case a software update is not released for a long period of time, this means that the adversary has almost no limitations in time. Furthermore, the attestation is performed only when the vehicle is not in the move and therefore it means that the vehicle is capable to drive without performing attestation of its ECUs, which in turn put the car in a non-safe position. In addition to this, since the manufacturer sends updates to each ECU which is needed to be updated and the attestation is performed by the ECU itself, this means that the ECUs inside the vehicle must be capable to establish a connection with the manufacturer and be powerful enough to be able to attest themselves, which can lead to big costs.

6.1.3 External attestation approach

A device attestation protocol can either be performed internally or externally. Performing some of the attestation processing externally can help to move some computationally expensive tasks out of the vehicle. Thus, less resources are needed inside each vehicle which leads to lower production costs. Another advantage with this approach is that the owner of the servers, for example, the vehicle manufacturer, can get an immediate answer whether a vehicle is compromised or not. This means that countermeasures can be put in place quickly if a compromised vehicle is detected. On the other hand, the external approach relies on infrastructure outside the vehicle which consists of roadside units (e.g. mobile networks), edge servers and trusted authorities. If such an infrastructure does not already exist globally, this means that the approach cannot be applied on the cars unless the respective infrastructure is installed. Also, the installation of RSU globally will lead to big costs. However, currently there is a big number of mobile networks which can be used to adopt the external attestation approach. In this case, it means that the RSUs need to cover the whole area where the vehicle can potentially drive and if it does not, the vehicle will be in a vulnerable position, since the attestation will not be performed without the presence of the constant signal with the RSU. In addition to this, RSUs today do not cover all areas and places like mountains, forests and deserts which can have

a bad connection. This implies that the adversary can use the vulnerability of the vehicle being located in a low-signal area and try to attack the vehicle. The external approach would also lead to a huge amount of communication data which could overwhelm the network. What is important with an approach like this is to preserve user integrity and not share information about the car owner or the location of the car. This is something to always keep in mind during attestation development.

6.1.4 Hardware attestation approach

Hardware-based attestation protocols are often associated with the high cost and as explained in Section 2.4, expensive hardware should be avoided in modern vehicles in order to not set a high price on the vehicle. Some examples of expensive resources are TPM, TrustedZone and Intel SGX which are used to store cryptographic keys, secret information, etc. This provides a high level of security and helps to achieve confidentiality and integrity. TPMs are specifically produced to solve the security issues in embedded systems and there are several attestation protocols that make use of them, for example ReDAS [30], [38], [51], [67] and [37].

The security aspect is the strength of hardware attestation methods, but as the name implies it makes use of extra hardware components. Specific cryptographic components such as TPMs are often quite expensive, thus less usable in low-cost systems. Enabling such hardware in all vehicle's devices (or at least in a major part of them) will increase the level of security inside the vehicle, but at the same time it will increase the price of the vehicle, this can potentially lead to a decrease in a number of sold vehicles. Usage of TPM or a kind of "SGX" is not wrong, but they are not suitable for attestation of in-vehicle networks, since they are designed for more powerful systems such as phones, laptops, etc.

There exist different degrees of hardware attestation and also less expensive hardware solutions. By different degrees could mean that expensive hardware is used not in all ECUs but in one place. That could for example be at a more powerful ECU. This would significantly lower production cost since it would limit the use of expensive hardware from 100 to 1 place. The other solution, where less expensive or already existing hardware is used could for example be the use of a PUF. This technique was explained in Section 2.3.2 and 4.6 and leverages the sometimes already existing hardware component SRAM.

6.1.5 Software attestation approach

In contrast to the hardware-based approach, the software approach requires no modification to the existing hardware since it only relies on computational times. This can be a big advantage, especially in low-end systems where there is no room for expensive components. Software approaches are good in some cases such as in terms of cost, effectiveness and no hardware modification, but one downside is the vulnerability to physical attacks and theoretically even some remote attacks. This is because of the assumption that an adversary can not compute the correct checksum in any

faster way than the original attestation code. This assumes two things. Firstly, the attestation code is optimized and cannot be optimized any further. Secondly, no more powerful hardware can be added to speed up the computations. It can be hard to justify an approach with these assumptions from a security point of view, since it is very difficult to know if the code cannot be optimized or if there will be no other hardware that is more powerful than the current hardware.

SWATT can be considered the role model when it comes to software-based attestation and a number of protocols were created on the base of SWATT. This protocol was released in 2004 and after that, there have been various variations of it and ways of trying to improve it. For example, a protocol called SMATT, tried to lower the computational overhead at the verifier by instead of calculating the result and comparing it to the provers result, it just compared incoming results from different provers. This is a smart solution to the overhead problem of the SWATT protocol. There are some flaws to this approach though, for this to work all provers need to receive the same challenge simultaneously. That could be a security vulnerability since an adversary can successfully carry through a replay attack if it gets hold of the current challenge.

Protocol LRMA [69] is more suitable for smart-grid networks, which are completely different in comparison to CAN bus. However, a very good feature, which can be applicable to vehicles, is categorizing devices into risky and non-risky ones, where the first category is attested less and the second type is attested more often. Enabling this feature in ECUs can help to reduce overhead in the CAN bus and result in better performance. Such an approach (categorizing devices) is similar to ALE protocol where the ECUs were divided into critical and non-critical. Combining LRMA and ALE can result in an attestation method where more critical ECUs will be attested more often than non-critical ECUs, as well as dividing ECUs into risky and non-risky can also be a solution where more risky ECUs are attested more often. The VIPER protocol tries to eliminate the assumption that an adversary can add a more powerful component, proxy, to compute the checksum within the correct time window. This relies on communication delays and will help to mitigate proxy attacks, but it does not solve the problem where an adversary has physical access to the prover and adds a more powerful hardware component. VIPER is one step closer to how the hybrid attestation methods work, but still does not solve the case where the adversary has physical access. On the other hand, getting physical access to one of the vehicle's ECU is a very difficult task and almost impossible for a random adversary, therefore the risk of a physical attack is not very high.

A centralised (master) protocol SCAPI and decentralized protocol EAPA are two schemes created in order to eliminate the risks of physical attacks. The prior protocol implicates the usage of a master device which constantly updates the session key at every device in the network and checks if the device has been non-active (physically tampered) by looking at the validity of the device's current session key. In case of invalidity, the tampered device is excluded from the network. The later protocol works in a similar way as SCAPI, but instead of the central device, the

neighboring devices send a heartbeat to each other and therefore can detect a physically tampered device. Having a feature of a heartbeat, which periodically will be exchanged by the devices or between master and simple ECUs, can be useful in the vehicle's CAN bus. This will make sure that the devices are "alive" and have not been physically attacked.

6.1.6 Hybrid attestation

In the hybrid attestation approaches the main purpose is to try to eliminate the assumptions from the software-based approaches. It is achieved by leveraging some kind of hardware component in the checksum computation. This does not need to be an expensive TPM, instead it could be a PUF which already can be applied on some low-end devices which got a SRAM chip. Thus, hybrid solutions achieve significantly higher security than software solutions, while still being affordable and applicable to low-end devices. AAoT is a promising hybrid attestation method for low-end devices since it uses the PUF efficiently and brings down the computational overhead compared to other hybrid protocols like PUFAtt and ATT-Auth. Overall, it can be said that the hybrid solution has a good proposition on price/security relation by eliminating the expensiveness of the hardware approach and the assumptions of the software approach.

6.1.7 Dynamic attestation approach

Dynamic attestation is something that definitely should be considered in a complete attestation solution, besides "normal" (static) attestation. This is because it is equally important to certify correct execution and control flow as it is to have the correct firmware. To allow an adversary to achieve arbitrary code execution by modifying the control flow is a severe security vulnerability. Some good methods for this are DIAT [2], LO-FAT [18], SMART [18] and C-FLAT [1]. The disadvantage with these protocols right now is that they require a relatively powerful verifier to be able to verify that the control flow is correct. If this issue can be solved these methods have big potential. It could even be possible to implement them in combination with a static attestation protocol, thus saving some computational resources.

6.2 Proposed attestation methods

Protocol 1, described in Section 5.2.1, is suitable for the future centralized architecture of vehicles. It fulfills the vehicle-specific requirements of low cost since it uses a minimal amount of hardware components at the same time as it is efficient. What it lacks is the resistance against some physical attacks. VIPER protects against some physical attacks where extra hardware components are added, but it does not protect against switching out a component or increasing the clock frequency of the processor. The purpose of Protocol 1 is mainly to eliminate remote attacks which AAoT and VIPER do successfully. It is also much more cost-effective compared to, for example, a decentralized approach, but instead the system relies completely on

a powerful master which creates a risk for a single point of failure.

Protocol 2, described in Section 5.2.2, was designed with some assumptions about the hardware components inside the vehicle. For example, it assumes that there exists a master ECU and a number of half master ECUs. The use of half-masters helps to adapt the protocol to the new architecture of the in-vehicle network which consists of one central processor and few gateways, as described in Section 2.1. Protocol 2 has some advantages over Protocol 1. Firstly, it distributes the computations to the half master ECUs which puts less effort on the master ECU. This also forms a more decentralized approach, which eliminates the single point of failure. Secondly, it constantly sends out heartbeats to the ECUs to detect physical attacks such as switching out or adding components. A decentralized approach like this works more efficiently since the half master ECUs can perform attestation in parallel. Also, Protocol 2 is based on ALE, which means that the response part of the protocol is using two nonces: old nonce (N_b) is used when creating the response key, as well as a new nonce (N) which on each attestation session will update the old nonce. Such an approach eliminates the possibility of replay attacks. In addition to this, Protocol 2 makes use of SMART which eliminates runtime attacks.

Both half-masters and critical ECUs are equipped with SRAM chips which adds an additional layer of security and makes it possible to use PUF and hybrid attestation, meanwhile non-critical ECUs are only using software attestation. It can be said that the usage of SRAM in critical ECUs and half-masters is a good trade-off between cost and security. SRAM also helps to make the attestation key of the half-master unreadable from the software and therefore more secure. This is a key difference between Protocol 2 and ALE - Protocol 2 is using PUF to create prover's attestation key, meanwhile ALE has attestation key in its secure storage. Using PUF in this case gives Protocol 2 a clear advantage over ALE in terms of resources and security, because there is no need to save the key in the secure storage as well as to not have the secure storage at all on the device, and the attestation key will be unique on every session depending on the nonce the prover gets from the verifier. As mentioned in the feasibility Section 5.3, an HSM could be used as a replacement for a PUF if that is already implemented on a certain ECU. An HSM would be equivalent in terms of features compared to a PUF so in this case the replacement would work.

Another advantage of Protocol 2 is the use of attestation rate, where devices that failed in attestation more times, are attested more frequently together with critical ECUs, meanwhile the device with low failure rate are attested less often. This can reduce computational power from non-problematic ECUs and focus more on problematic ones. This allows monitoring of problematic ECUs more often. Since critical ECUs' security is important for the vehicle's and driver's safety, a constant and frequent attestation of them is also very important. Last, but not least, making software updates in a secure manner by eliminating the vulnerability when the new software is in RAM, was considered in the protocol and makes the updates more secure. The only drawback with Protocol 2 is that it can require some modification

to the existing hardware, such as adding SRAM to critical ECUs, thus leading to a higher cost. But, in the future we are moving towards an IVN consisting of less and more powerful ECUs which is optimal for this protocol.

All feedback given to the Protocol 1 and Protocol 2 from the industry experts, as described in Section 5.3, need to be taken into account. Increasing a timeframe for response messages in VIPER/AAoT as well as assigning priority to the messages participating in the attestation can potentially solve the problem with jitters. It is also important to give all the verifiers additional storage which would be enough to save all information about the provers. Taking into account false positives during the heartbeats sending in Protocol 2 should be taken into account.

Both Protocol 1 and Protocol 2 are able of preventing remote attacks where software modification is possible, therefore both protocols are resistible to the adversary from adversary model 1, because both VIPER and AAoT successfully detect code modification. As for adversary model 2, which implicates physical access to the ECUs when the hardware can be replaced, Protocol 1 is not able to resist this type of adversary. However, Protocol 2 has the Heartbeat feature which can detect the absence of a device and therefore this protocol is capable to resist the adversary from the adversary model 2.

As a summary of the discussion over the proposed protocols, it can be said that both protocols are based on the methods described in Section 4 and each of the protocols combines different methodologies of attestation in order to meet the requirements for both attestation and the vehicle industry.

6.3 What to do if attestation fails?

The importance of attestation is undoubtedly significant for a secure driving experience. However, the natural question arises about the next actions that should be taken after a failed attestation. One simple approach would be to block the vehicle completely and not allow it to drive. At the same time, what would happen if the driver is in an emergency situation when he/she needs to drive and because of some error in one of the devices, he/she is not able to drive? On the one hand, this approach can guarantee that the vehicle will not put the driver and road users in danger, but on the other hand, if it is a non-important device, like the AC, which failed the attestation, why should it block the driving capability? Proceeding from this reasoning, we would propose to block the vehicle's driving capability only in the cases when the most important and the most critical devices' firmware was not approved during the attestation procedure during the vehicle's startup/unlocking. However, an alert for the driver, even if a non-critical ECU failed the attestation, should be shown on the screen in order to make the driver aware of it.

A more complicated case would be if the attestation failed during the movement. Again, if a non-critical device has not been approved this would not have a major impact on the driver's safety, but if a critical device, such as the braking system

would be affected, this can create a danger. In the latter case, if the vehicle's ability to drive would be blocked right away during the movement, this will definitely be a danger both for the driver and for other road users. Therefore, an immediate break would not be a good solution. We propose in this case to, firstly, show an alert about failed attestation and ask the driver to stop somewhere and secondly, as a last measure, gradually decrease the speed of the vehicle in order to make the driver stop. This will not create a danger, but a slow deceleration will make the movement safer and finally stop the vehicle.

6.4 Trends in device attestation

A trend that was trivial to see when conducting the literature review was the ambition to move away from expensive hardware such as TPMs. This trend was mostly aimed at low-end devices which also is what this literature survey is. Researchers try to find alternative solutions in form of either hybrid or software attestation methods. This is done to achieve efficient and low-cost solutions. Another trend we found, by collaborating with engineers at Volvo Cars, is that the industry is moving from using a big number of ECUs to one more powerful ECU.

6.5 Alternative solutions for the future

Over the past years, there was an increase in the amount of ECUs inside vehicles, reaching numbers over 100. Some companies such as Intel and Nvidia are today working towards fewer and more powerful ECUs inside vehicles [26]. But what does this mean for the vehicles? This can provide several benefits, for example, lower costs, less weight, future proof, faster time to market and higher security. If we consider this possible scenario to take place in the future, it would mean that the modern device attestation methods will have to change to adapt to the new reality. For example, if the vehicle has only one single ECU with big computational power that is capable to do all tasks inside the vehicle, the decentralized approach becomes irrelevant. The software-based attestation methods would not be that suitable either since if there exists a powerful ECU with a high-end processor and perhaps even a graphic card then it would be more suitable to go for some kind of hardware attestation method. As previously mentioned in the hardware attestation Section 4.4, ARM's TrustedZone and Intel's Intel Software Guard Extensions (SGX) provide hardware security and these features often exist in modern high-end CPUs. Having some type of SGX or TrustedZone in the scenario described above could be a justifiable solution for making attestation secure.

By the above, it is meant that a hardware attestation method would probably be a better way to go in the future if the vehicle industry goes towards less but more powerful ECUs. Depending on the amount of ECUs that will exist in a vehicle, different solutions could be suitable. For example, if there is just one powerful ECU then a hardware attestation solution is preferable but, if there are 5-10 relatively powerful ECUs then perhaps a hybrid attestation solution is a better choice since

that is a more cost-efficient solution. Even a decentralized approach can potentially work when there will be some limited number of powerful ECUs, which can attest themselves and other simple ECUs as well. This would eliminate the problem of the single point of failure.

With the rise of IoT and 5G, the mobile network will only grow in the future. This opens great opportunities for RSU infrastructure development. If the mobile network will cover most of the roads in the world, this could transform the modern attestation method in a way that the attestation will be more and more external and all the processes will happen in the cloud/remote server. Such transformation has obvious benefits: all computational power is moved to the cloud and the vehicle does not need powerful and costly ECUs, which results in less production price as well as weight. Also, considering that cellular internet is becoming faster and faster, the communication between the vehicle and remote the server can be established more efficiently. However, this would mean that security between these two objects must be fully fulfilled.

7
Conclusion & Future work

Embedded systems have become vital parts of modern vehicles and today they are inseparable. This implies a more software-driven vehicle, where critical functionality relies on correct code. The need to verify this software, i.e. performing device attestation, has not been greater than it is today. In this paper, we present a survey on the state of the art when it comes to device attestation. Furthermore, this work categorizes and discusses different attestation methods in relation to vehicle-specific challenges. Based on this work, we have proposed our own methods that aim to both fulfill the requirements of a secure attestation method and be suitable for the vehicle industry which requires low-cost, efficient and secure solutions. Our proposed methods are designed with the idea to work today and in the near future. We also discuss hypothetical solutions for the non-immediate future, since the vehicle industry is making rapid advancements and is in constant movement.

As mentioned in the Limitations, Section 3.3, time was a limiting factor in this project. Because of this, the implementation and experiments of the state-of-the-art device attestation methods and our own methods were left out. This work would be a good starting point for a future project to evaluate the performance of different attestation methods. The optimal scenario would be if the described attestation methods, as well as proposed protocols, could be tried out and evaluated on a real IVN with the appropriate ECU hardware. Since this is an upcoming and rapidly growing research area it would also be possible to extend this survey with new attestation methods in the future. Perhaps, there will be extra categories added to our proposed ones. Another subject to investigate would be how much the current attestation methods modify the IVN in form of hardware changes and also look into the cost of these changes. This could be interesting and usable information for the vehicle manufacturers since they often strive for low-cost solutions. Further, it could be a good deciding factor when it comes down to choosing between methods to implement.

Bibliography

[1] Tigist Abera et al. *C-FLAT: Control-FLow ATtestation for Embedded Systems Software*. 2016. arXiv: 1605.07763 [cs.CR].

[2] Tigist Abera et al. "DIAT: Data Integrity Attestation for Resilient Collaboration of Autonomous Systems". In: *Proceedings 2019 Network and Distributed System Security Symposium* (2019).

[3] Tejasvi Alladi et al. "A Lightweight Authentication and Attestation Scheme for In-Transit Vehicles in IoV Scenario". In: *IEEE Transactions on Vehicular Technology* 69.12 (2020), pp. 14188–14197. DOI: 10.1109/TVT.2020.3038834.

[4] Muhammad Naveed Aman and Biplab Sikdar. "ATT-Auth: A Hybrid Protocol for Industrial IoT Attestation With Authentication". In: *IEEE Internet of Things Journal* 5.6 (2018), pp. 5119–5131. DOI: 10.1109/JIOT.2018.2866623.

[5] Muhammad Naveed Aman et al. "HAtt: Hybrid Remote Attestation for the Internet of Things With High Availability". In: *IEEE Internet of Things Journal* 7.8 (2020), pp. 7220–7233. DOI: 10.1109/JIOT.2020.2983655.

[6] Moreno Ambrosin et al. "Collective Remote Attestation at the Internet of Things Scale: State-of-the-Art and Future Challenges". In: *IEEE Communications Surveys Tutorials* 22.4 (2020), pp. 2447–2461. DOI: 10.1109/COMST.2020.3008879.

[7] Moreno Ambrosin et al. "SANA: Secure and Scalable Aggregate Network Attestation". In: Oct. 2016, pp. 731–742. DOI: 10.1145/2976749.2978335.

[8] Moreno Ambrosin et al. "Toward Secure and Efficient Attestation for Highly Dynamic Swarms: Poster". In: *Proceedings of the 10th ACM Conference on Security and Privacy in Wireless and Mobile Networks*. WiSec '17. Boston, Massachusetts: Association for Computing Machinery, 2017, pp. 281–282. ISBN: 9781450350846. DOI: 10.1145/3098243.3106026. URL: https://doi.org/10.1145/3098243.3106026.

[9] Mahmoud Ammar et al. "SlimIoT: Scalable Lightweight Attestation Protocol for the Internet of Things". In: *2018 IEEE Conference on Dependable and Secure Computing (DSC)*. 2018, pp. 1–8. DOI: 10.1109/DESEC.2018.8625142.

[10] Ross Anderson et al. "A New Family of Authentication Protocols". In: *Operating Systems Review (ACM)* 32 (Jan. 1999). DOI: 10.1145/302350.302353.

[11] Orlando Arias et al. "Device attestation: Past, present, and future". In: *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*. 2018, pp. 473–478. DOI: 10.23919/DATE.2018.8342055.

[12] Frederik Armknecht et al. "A Security Framework for the Analysis and Design of Software Attestation". In: *Proceedings of the 2013 ACM SIGSAC Conference on Computer amp; Communications Security*. CCS '13. Berlin, Germany:

Association for Computing Machinery, 2013, pp. 1–12. ISBN: 9781450324779. DOI: 10.1145/2508859.2516650. URL: https://doi.org/10.1145/2508859.2516650.

[13] N. Asokan et al. "SEDA: Scalable Embedded Device Attestation". In: Oct. 2015, pp. 964–975. DOI: 10.1145/2810103.2813670.

[14] Jill Britton. *What is ISO21434? Compliance Tips for Automotive Software Developers.* URL: https://www.perforce.com/blog/qac/ISO-21434-Compliance.

[15] Xavier Carpent et al. "Lightweight Swarm Attestation: A Tale of Two LISA-s". In: *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security* (2017).

[16] Claude Castelluccia et al. "On the Difficulty of Software-Based Attestation of Embedded Devices". In: *Proceedings of the 16th ACM Conference on Computer and Communications Security*. CCS '09. Chicago, Illinois, USA: Association for Computing Machinery, 2009, pp. 400–409. ISBN: 9781605588940. DOI: 10.1145/1653662.1653711. URL: https://doi.org/10.1145/1653662.1653711.

[17] Brett Daniel. *What Is Intel SGX?* Jan. 2021. URL: https://www.trentonsystems.com/blog/what-is-intel-sgx.

[18] Ghada Dessouky et al. "LO-FAT". In: *Proceedings of the 54th Annual Design Automation Conference 2017* (June 2017). DOI: 10.1145/3061639.3062276. URL: http://dx.doi.org/10.1145/3061639.3062276.

[19] Karim ElDefrawy, Norrathep Rattanavipanon, and Gene Tsudik. *HYDRA: HYbrid Design for Remote Attestation (Using a Formally Verified Microkernel)*. 2017. arXiv: 1703.02688 [cs.CR].

[20] Tsudik Gene Eldefrawy Karim Perito Daniele. "SMART: Secure and Minimal Architecture for (Establishing a Dynamic) Root of Trust". In: (Jan. 2012).

[21] Jason England. *Where to Buy a TPM 2.0 for Windows 11.* Mar. 2022. URL: https://www.tomshardware.com/news/where-to-buy-tpm-2.0-for-windows-11.

[22] Wei Feng et al. "AAoT: Lightweight Attestation and Authentication of low-resource Things in IoT and CPS". In: *Computer Networks* 134 (Feb. 2018). DOI: 10.1016/j.comnet.2018.01.039.

[23] Aurélien Francillon et al. "A minimalist approach to Remote Attestation". In: *2014 Design, Automation Test in Europe Conference Exhibition (DATE)*. 2014, pp. 1–6. DOI: 10.7873/DATE.2014.257.

[24] Jason Hughes. *The Big Tesla Hack: A hacker gained control over the entire fleet, but fortunately he's a good guy.* Mar. 2017. URL: https://electrek.co/2020/08/27/tesla-hack-control-over-entire-fleet/.

[25] Ahmad Ibrahim, Ahmad-Reza Sadeghi, and Gene Tsudik. "US-AID: Unattended Scalable Attestation of IoT Devices". In: *2018 IEEE 37th Symposium on Reliable Distributed Systems (SRDS)*. 2018, pp. 21–30. DOI: 10.1109/SRDS.2018.00013.

[26] Intel. *ECU Consolidation Reduces Vehicle Cost, Weight, and Testing.* URL: https://www.intel.com/content/dam/www/public/us/en/documents/white-papers/ecu-consolidation-white-paper.pdf.

[27] ISO. *ISO/IEC Information technology — Trusted Platform Module — Part 1: Overview*. May 2009. URL: https://www.iso.org/standard/50970.html.

[28] Stefan Katzenbeisser et al. "PUFs: Myth, Fact or Busted? A Security Evaluation of Physically Unclonable Functions (PUFs) Cast in Silicon". In: *Cryptographic Hardware and Embedded Systems – CHES 2012*. Ed. by Emmanuel Prouff and Patrick Schaumont. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 283–301. ISBN: 978-3-642-33027-8.

[29] Mohammed Khodari et al. "Decentralized Firmware Attestation for In-Vehicle Networks". In: *Proceedings of the 5th on Cyber-Physical System Security Workshop*. CPSS '19. Auckland, New Zealand: Association for Computing Machinery, 2019, pp. 47–56. ISBN: 9781450367875. DOI: 10.1145/3327961.3329529. URL: https://doi.org/10.1145/3327961.3329529.

[30] Chongkyung Kil et al. "Remote attestation to dynamic system properties: Towards providing complete system integrity evidence". In: *2009 IEEE/IFIP International Conference on Dependable Systems Networks*. 2009, pp. 115–124. DOI: 10.1109/DSN.2009.5270348.

[31] Kyounggon Kim et al. "Cybersecurity for autonomous vehicles: Review of attacks and defense". In: *Computers Security* 103 (2021), p. 102150. ISSN: 0167-4048. DOI: https://doi.org/10.1016/j.cose.2020.102150. URL: https://www.sciencedirect.com/science/article/pii/S0167404820304235.

[32] Hubert Kirrmann. "Fault Tolerant Computing in Industrial Automation". In: *Switzerland: ABB Research Center* (Mar. 2015).

[33] Florian Kohnhäuser. "Advanced Remote Device Attestation Protocols for Embedded systems". English. PhD thesis. Darmstadt, Germany, 2019.

[34] 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.

[35] Joonho Kong et al. "PUFatt: Embedded platform attestation based on novel processor-based PUFs". In: *2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC)*. 2014, pp. 1–6. DOI: 10.1145/2593069.2593192.

[36] Karl Koscher et al. "Experimental Security Analysis of a Modern Automobile". In: *2010 IEEE Symposium on Security and Privacy*. 2010, pp. 447–462. DOI: 10.1109/SP.2010.34.

[37] Kari Kostiainen et al. "Key Attestation from Trusted Execution Environments". In: June 2010, pp. 30–46. ISBN: 978-3-642-13868-3. DOI: 10.1007/978-3-642-13869-0_3.

[38] Xeno Kovah et al. "New Results for Timing-Based Attestation". In: *2012 IEEE Symposium on Security and Privacy*. 2012, pp. 239–253. DOI: 10.1109/SP.2012.45.

[39] Boyu Kuang et al. "A survey of remote attestation in Internet of Things: Attacks, countermeasures, and prospects". In: *Computers Security* 112 (2022), p. 102498. ISSN: 0167-4048. DOI: https://doi.org/10.1016/j.cose.2021.102498. URL: https://www.sciencedirect.com/science/article/pii/S0167404821003229.

[40] Tencent Keen Security Lab. “Experimental Security Assessment of BMW Cars: A Summary Report”. In: (May 2018).

[41] Fred Lambert. *Tesla fights back against owners hacking their cars to unlock performance boost*. Aug. 2020. URL: https://electrek.co/2020/08/22/tesla-fights-back-against-owners-hacking-unlock-performance-boost/.

[42] Gyesik Lee et al. “Formally verifiable features in embedded vehicular security systems”. In: *2009 IEEE Vehicular Networking Conference (VNC)*. 2009, pp. 1–7. DOI: 10.1109/VNC.2009.5416378.

[43] Yanlin Li, Jonathan M. McCune, and Adrian Perrig. “SBAP: Software-Based Attestation for Peripherals”. In: *Trust and Trustworthy Computing*. Ed. by Alessandro Acquisti, Sean W. Smith, and Ahmad-Reza Sadeghi. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 16–29. ISBN: 978-3-642-13869-0.

[44] Yanlin Li, Jonathan M. McCune, and Adrian Perrig. “VIPER: Verifying the Integrity of PERipherals’ Firmware”. In: *Proceedings of the 18th ACM Conference on Computer and Communications Security*. CCS ’11. Chicago, Illinois, USA: Association for Computing Machinery, 2011, pp. 3–16. ISBN: 9781450309486. DOI: 10.1145/2046707.2046711. URL: https://doi.org/10.1145/2046707.2046711.

[45] Dr. Charlie Miller and Chris Valasek. “Remote Exploitation of an Unaltered Passenger Vehicle”. In: *Black Hat USA* (Aug. 2015).

[46] Dennis K. Nilsson, Lei Sun, and Tatsuo Nakajima. “A Framework for Self-Verification of Firmware Updates over the Air in Vehicle ECUs”. In: *2008 IEEE Globecom Workshops*. 2008, pp. 1–5. DOI: 10.1109/GLOCOMW.2008.ECP.56.

[47] Ivan De Oliveira Nunes et al. “VRASED: A Verified Hardware/Software Co-Design for Remote Attestation”. In: *28th USENIX Security Symposium (USENIX Security 19)*. Santa Clara, CA: USENIX Association, Aug. 2019, pp. 1429–1446. ISBN: 978-1-939133-06-9. URL: https://www.usenix.org/conference/usenixsecurity19/presentation/de-oliveira-nunes.

[48] Hisashi Oguma et al. “New Attestation Based Security Architecture for In-Vehicle Communication”. In: *IEEE GLOBECOM 2008 - 2008 IEEE Global Telecommunications Conference*. 2008, pp. 1–6. DOI: 10.1109/GLOCOM.2008.ECP.369.

[49] Haemin Park et al. “SMATT: Smart Meter ATTestation Using Multiple Target Selection and Copy-proof Memory”. In: vol. 203. Nov. 2012. ISBN: 978-94-007-5698-4. DOI: 10.1007/978-94-007-5699-1_90.

[50] Zhou QIN et al. “New ECU Attestation and Encryption Mechanism for In-Vehicle Communication”. In: *DEStech Transactions on Engineering and Technology Research* (Nov. 2016). DOI: 10.12783/dtetr/ssme-ist2016/3987.

[51] Dries Schellekens, Brecht Wyseur, and Bart Preneel. “Remote Attestation on Legacy Operating Systems with Trusted Platform Modules”. In: *Sci. Comput. Program.* 74.1–2 (Dec. 2008), pp. 13–22. ISSN: 0167-6423. DOI: 10.1016/j.scico.2008.09.005. URL: https://doi.org/10.1016/j.scico.2008.09.005.

[52] Geert-Jan Schrijen and Vincent van der Leest. "Comparative analysis of SRAM memories used as PUF primitives". In: *2012 Design, Automation Test in Europe Conference Exhibition (DATE)*. 2012, pp. 1319–1324. DOI: 10.1109/DATE.2012.6176696.

[53] Steffen Schulz, Ahmad-Reza Sadeghi, and Christian Wachsmann. "Short paper: Lightweight remote attestation using physical functions". In: Jan. 2011, pp. 109–114. DOI: 10.1145/1998412.1998432.

[54] A. Seshadri et al. "SWATT: softWare-based attestation for embedded devices". In: *IEEE Symposium on Security and Privacy, 2004. Proceedings. 2004.* 2004, pp. 272–282. DOI: 10.1109/SECPRI.2004.1301329.

[55] Arvind Seshadri, Mark Luk, and Adrian Perrig. "SAKE: Software Attestation for Key Establishment in Sensor Networks". In: *Distributed Computing in Sensor Systems*. Ed. by Sotiris E. Nikoletseas et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 372–385. ISBN: 978-3-540-69170-9.

[56] Arvind Seshadri et al. "SCUBA: Secure Code Update by Attestation in sensor networks". In: vol. 2006. Jan. 2006, pp. 85–94. DOI: 10.1145/1161289.1161306.

[57] Ioannis Sfyrakis and Thomas Gross. *A Survey on Hardware Approaches for Remote Attestation in Network Infrastructures*. 2020. arXiv: 2005.12453 [cs.CR].

[58] Rodrigo Vieira Steiner and Emil Lupu. "Attestation in Wireless Sensor Networks: A Survey". In: *ACM Comput. Surv.* 49.3 (Sept. 2016). ISSN: 0360-0300. DOI: 10.1145/2988546. URL: https://doi.org/10.1145/2988546.

[59] Ivan Studnia et al. "Survey on security threats and protection mechanisms in embedded automotive networks". In: *2013 43rd Annual IEEE/IFIP Conference on Dependable Systems and Networks Workshop (DSN-W)*. 2013, pp. 1–12. DOI: 10.1109/DSNW.2013.6615528.

[60] *The ECU killer: broken authentication in automotive security*. URL: https://cydrill.com/cyber-security/the-unlikely-ecu-killer-broken-authentication-in-automotive-security/.

[61] Vrizlynn L.L. Thing and Jiaxi Wu. "Autonomous Vehicle Security: A Taxonomy of Attacks and Defences". In: *2016 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*. 2016, pp. 164–170. DOI: 10.1109/iThings-GreenCom-CPSCom-SmartData.2016.52.

[62] SCOTT THORNTON. *Arm TrustZone explained*. Dec. 2017. URL: https://www.microcontrollertips.com/embedded-security-brief-arm-trustzone-explained/.

[63] UN. *Three landmark UN vehicle regulations enter into force*. Feb. 2021. URL: https://unece.org/sustainable-development/press/three-landmark-un-vehicle-regulations-enter-force.

[64] UN. *UN Regulation No. 155 - Cyber security and cyber security management system*. Mar. 2021. URL: https://unece.org/transport/documents/2021/03/standards/un-regulation-no-155-cyber-security-and-cyber-security.

[65] *UNECE Vehicle Regulation for Cyber Security Software Updates*. Mar. 2021. URL: https://conti-engineering.com/unece-vehicle-regulation-for-cyber-security-software-updates/.

[66] Alessandro Visintin et al. $SAFE^d$: $Self-AttestationForNetworksofHeterogeneousEmbedde$ Sept. 2019.

[67] Jorden Whitefield et al. *Privacy-Enhanced Capabilities for VANETs using Direct Anonymous Attestation*. 2017.

[68] Wenwen Yan et al. "EAPA: Efficient Attestation Resilient to Physical Attacks for IoT Devices". In: *Proceedings of the 2nd International ACM Workshop on Security and Privacy for the Internet-of-Things*. IoT Samp;P'19. London, United Kingdom: Association for Computing Machinery, 2019, pp. 2–7. ISBN: 9781450368384. DOI: 10.1145/3338507.3358614. URL: https://doi.org/10.1145/3338507.3358614.

[69] Xinyu Yang et al. "Towards a Low-Cost Remote Memory Attestation for the Smart Grid". In: *Sensors (Basel, Switzerland)* 15 (Aug. 2015), pp. 20799–20824. DOI: 10.3390/s150820799.

[70] Ezer Osei Yeboah-Boateng. "Cyber-Security Challenges with SMEs in Developing Economies: Issues of Confidentiality, Integrity Availability (CIA)". English. PhD thesis. Denmark, 2013. ISBN: 978-87-7152-038-5.