Ensuring Trustworthy Automated Road Vehicles: A Software Integrity Validation Approach

Dominik Püllen*, Felix Klement†, Alexey Vinel‡, Stefan Katzenbeisser§
University of Passau, Germany*†§
{dominik.puellen, felix.klement, stefan.katzenbeisser}@uni-passau.de
Karlsruhe Institute of Technology, Germany‡
alexey.vinel@kit.edu

***Abstract*—This paper presents a cascading scheme designed to measure and validate the software integrity of interconnected and automated road vehicles. With modern vehicular architectures decoupling hardware from specific functions and supporting frequent updates, we require proactive means to detect software manipulations, especially as compromised control units pose a severe risk to passenger safety. Therefore, we must ensure the benignity of critical software components that impact the vehicle's dynamics while allowing for rapid adaptions and user customizations. Our scheme requires vehicles to obtain proof of trustworthiness before performing specific actions, such as driving autonomously or joining a vehicle fleet. To achieve this, we employ remote attestation techniques to compute an integrity identifier verified and validated by a trusted third party. We specifically design the process of integrity measurement for hierarchically organized networks of control units, as is the case in domain-based and zonal architectures. For evaluation, we utilize a recently presented zonal architecture deployed in four fully automated road vehicles and demonstrate that our scheme incurs an acceptable overhead.**

***Index Terms*—software integrity, automotive security**

I. Introduction

The automatization and interconnection of road vehicles have significantly boosted the importance of software and caused profound technological changes in the automotive domain. Historically, automobiles have relied on numerous Electronic Control Units (ECUs) to perform dedicated tasks, communicating through specialized automotive protocols such as the Controller Area Network (CAN) [1]. In contrast, modern software-defined vehicles employ centralized Ethernet-based Electrical/Electronic (E/E) architectures, featuring fewer but more powerful ECUs that serve multiple purposes. This software-driven approach allows for updates, the dynamic loading of functions, and the distributed software development independent of the underlying hardware. Adopting a service-oriented communication approach is crucial in developing modular, scalable, and interoperable software. Prominent examples include the SOME/IP [2] middleware and the Data Distribution Service (DDS), both integrated into the AUTOSAR Adaptive Platform [3], a software architecture developed by leading automotive stakeholders. The latest effort in this regard is the Automotive Service-Oriented software Architecture (ASOA) [4] middleware, aiming at creating high-performance and scalable automotive software. While the software-driven approach undoubtedly paves the way for vehicles with Advanced Driving Assistance Systems (ADASs) and autonomous driving capabilities, the openness and updatability of such systems lead to significant security concerns [5]. Researchers have demonstrated how even approved safe automobiles become deadly traps if attackers successfully compromise them [6], [7]. As such, regulators have called for a holistic security view throughout the entire vehicle lifecycle [8]. In response, the industry has adapted its organizational structures, processes, and standardized security engineering frameworks [9]. By now, there are decent solutions to protect in-vehicle communication from manipulation attacks. However, the attacker may also reside on the ECUs, especially in an updatable system with a frequently changing software base. The recently published UN Regulation No. 156 [10] highlights the importance of protecting software from manipulations as compromised ECUs may have devastating safety consequences.

To address this challenge, we propose a proactive scheme that determines the vehicle's integrity whenever the vehicle starts, or software is updated from a remote source. Our scheme computes an integrity identifier that incorporates the software state of each ECU and which is verified and validated by a Trusted Software Authority (TSA). Verification refers to cryptographic authenticity and correctness, while validation involves interpreting the identifier and comparing it to prevailing rules. The last step is necessary because software-defined vehicles may permit passengers to intentionally deploy their applications, such as a customized infotainment system. Hence, depending on the legal requirements, the validation process can consider a vehicle trustworthy, even if *uncritical* software has not been reviewed and, thus, is unknown to the TSA. If the vehicle is considered trustworthy, a certificate is issued, allowing the vehicle to register for further actions such as automated driving. This way, untrusted vehicles can be excluded from traffic before they cause any harm.

We use hardware isolation techniques to compute the aforementioned integrity identifier in a cascading fashion, making our approach specifically suitable for hierarchically organized architectures, where some ECUs may be shielded for safety or technical reasons. We have implemented and evaluated the scheme for a recently presented zonal architecture and simulated it in two different ETSI scenarios. Our work emphasizes the significance of ensuring software integrity in modern vehicles and contributes to ongoing efforts to develop secure and safe vehicular architectures.

II. Related Work

Remote Attestation. The principles of remote attestation have been widely discussed [11]. Typically, a *prover* makes a claim about its system to a remote party, also referred to as the *verifier*. During attestation, the prover provides computational evidence about this claim to the verifier, who either confirms or rejects it. Special hardware extensions like a Trusted Platform Module (TPM), Intel Software Guard Extensions (SGX), or the ARM TrustZone provide an isolated execution environment in which the prover creates the evidence. Alternatively, lightweight solutions [12], [13] can be used for embedded systems. In our work, we leverage the technique of remote attestation to securely execute the measurement code.

Vehicle Integrity. Erickson et al. [14] propose autonomous vehicle contracts to enforce desired driving behavior within a platoon of self-driving vehicles. These contracts define a set of driving parameters, such as speed and safety distance that the vehicles in the platoon must adhere to. To ensure compliance with the contract, powertrain and brake commands are continuously monitored and filtered. The monitoring process takes place in a hardware-isolated trusted environment, or "enclave", which allows vehicles in the platoon to attest each other's monitoring code and ensure conformity with the driving contract. In contrast to Erickson et al.'s approach, our work does not focus on the integrity of specific commands during vehicle operation. Instead, we aim to confirm the integrity of a vehicle's software through a measurement process that we have postulated in earlier work [15], though without specifying a complete protocol. Kohnhäuser et al. [16] presented a remote attestation scheme to detect compromised software in ECUs at load time. However, their approach assumes a trusted verifier in each vehicle, which poses a challenge for incorporating regular software updates from remote sources.

In-Vehicle Network Security. In-Vehicle communication has been found to be particularly vulnerable to cyberattacks due to the absence of authentication mechanisms in widely-deployed automotive protocols. As a result, numerous *reactive* security measures have been developed to mitigate this issue. For instance, the AUTOSAR Secure Onboard Communication (SecOC) [17] is an established and standardized solution for traditional signal-based protocols. With regard to service-oriented frameworks, there are tailored security extensions such as those developed for SOME/IP by Zelle et al. [18] and Iorio et al. [19], as well as a standardized security model for DDS [20]. These solutions are doing great in preventing eavesdropping and network manipulation attacks. However, they lack a mechanism to detect an adversary on the ECUs, which we attempt to address in this work.

III. Background on E/E Architectures

The current trend towards dynamic and updatable automotive software accompanies the transition from distributed E/E architectures to centralized ones. In addition to the long-used distributed multi-master architecture, two centralized architectures have emerged: domain-based and zonal. As depicted in Figure 1, a domain-based architecture comprises multiple functional Domain Controllers (DCs) interconnected through a central gateway.

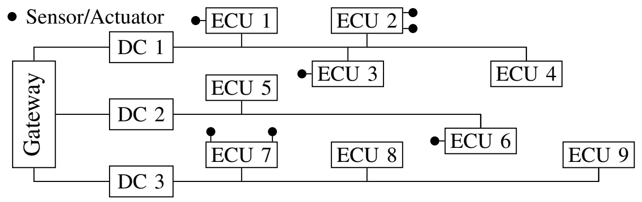

Fig. 1: In domain-based architectures, ECUs of a specific domain are physically close. Typically each performs a dedicated functional task related to the area of the respective domain. A domain has a Domain Controller (DC) that are interconnected through a central gateway.

A domain comprises ECUs responsible for specific functional tasks associated with the respective domain. Such architectures are most effective when the connected devices are in close proximity. Typical domains in automobiles include the powertrain, infotainment, and ADAS. In contrast, as shown in Figure 2, zonal architectures deploy even fewer, more powerful, and general-purpose ECUs, reducing the overall complexity and the number of potential failure points.

Zones are characterized by nearby devices that, unlike domains, often possess substantial computational resources and may execute completely unrelated operations. A central High Performance Controller (HPC) typically performs management tasks between the zones. Given the ability to support the needs of modular and decoupled software components, the automotive industry has increasingly turned towards zonal architectures. In both architecture types, ECUs are arranged hierarchically and may be linked to smaller peripheral devices like sensors and actuators. Such control units are only connected to specific ECUs and thus are shielded from the main bus, either for safety reasons or because of technical limitations. Our scheme attempts to consider even the integrity of shielded ECUs by working in a cascading way where each ECU includes the integrity measurement of its children into its identifier. In the following, we denote an ECU $\mathcal{E}_{\text{i}}$ a child of $\mathcal{E}_{\text{j}}$ if $\mathcal{E}_{\text{i}}$ is linked to $\mathcal{E}_{\text{j}}$ and located hierarchically below $\mathcal{E}_{\text{i}}$.

IV. Preliminaries

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

We introduce a generic Registration Unit (RU) requiring vehicles to register for actions they intend to perform. The idea behind the RU is to grant permission to the requested action only if the vehicle can provide proof of trustworthiness, thereby ensuring a validated software state, as compromise may pose a significant hazard to other traffic members. The precise role of the RU can be customized to suit a variety of mobility concepts. For instance, a vehicle may wish to participate in automated traffic in an urban environment, where the RU could comprise all automobiles in a specified perimeter that share environmental sensor data and, thus, need to trust each other. Alternatively, the RU could be a traffic surveillance unit, as has been suggested by recent research [21] and even required by a German draft law [22] from 2019. Another potential use case may be a platoon whose master vehicle requires proof of trustworthiness from vehicles seeking to join.

In our system, a vehicle's trustworthiness is attested by the Trusted Software Authority (TSA), another external component. It is primarily responsible for the maintenance of automotive software and, for that purpose, keeps a copy of each authentic software component. The TSA acts as a *verifier* and issues a certificate if the vehicle can provide evidence for its benign software state. Notably, unlike related work [16] that assumes an in-vehicle trust anchor, our approach entails each vehicle solely trusting the remote TSA and possessing its public key. Furthermore, we assume the existence of a shared identity key, denoted as $\mathsf{SK}_{\mathcal{E}_i}$, between each ECU and the TSA. We acknowledge that maintaining a separate key for each ECU in every vehicle does not scale in a large automotive ecosystem. We envision a key derivation mechanism, allowing the TSA to computationally determine $\mathsf{SK}_{\mathcal{E}_i}$ based on the vehicle identity *VID*. However, a practical solution is orthogonal to the problem of ensuring trustworthiness and is therefore left for future research.

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

V. Integrity Validation Scheme

Our scheme for validating the vehicle integrity comprises five consecutive steps, executed every time a proof of trustworthiness is required, typically when a vehicle starts. A trustworthy vehicle receives a certificate that enables it to register for an automotive action, such as participating in urban traffic or joining a platoon. Figure 2 illustrates our scheme schematically.

Step 1 (Initialization). The integrity measurement begins with the central ECU $\mathcal{E}_C$ sending an attestation request to the TSA, containing the vehicle's identity *VID*. In response, the TSA generates a random nonce $\mathcal{N}$ and a distinct attestation key $\mathcal{AK}$, which will be used to authenticate the integrity measurements. The purpose of $\mathcal{N}$ is to ensure freshness and thwart $\mathcal{A}dv$ from replaying out-dated measurements. To

Algorithm 1 Software Integrity Measurement

```
1:  procedure MEASURE($\mathsf{SC}_{\mathcal{E}_i}$, $\mathsf{IntID}^C_{\mathcal{E}_i}$, $\mathsf{IntM}^C_{\mathcal{E}_i}$, $\mathsf{C}^{\mathcal{AK}}_{\mathcal{E}_i}$, $\mathcal{N}$)
2:      $\mathsf{IntM} \leftarrow array()$                    ▷ contains all measurements
3:      $\mathcal{AK} \leftarrow Dec(\mathsf{C}^{\mathcal{AK}}_{\mathcal{E}_i})_{\mathsf{SK}_{\mathcal{E}_i}}$                ▷ get attestation key
4:      for each $\mathsf{sw} \in \mathsf{SC}_{\mathcal{E}_i}$ do               ▷ measure sw integrity
5:          $\mathtt{bin} = Load(\mathsf{sw})$
6:          $\mathtt{hash} \leftarrow Hash(\mathtt{bin})$        ▷ e.g., using Linux IMA
7:          $\mathsf{IntM}[\mathcal{E}_i].Append(\mathsf{sw}, \mathtt{hash})$
8:      end for
9:      $\mathtt{authboot} \leftarrow GetAuthBootCodes()$
10:     $\mathsf{IntM}[\mathcal{E}_i].Append(\text{“bootchain”}, \mathtt{authboot})$
11:     while $idx \neq \text{len}(\mathsf{IntID}^C_{\mathcal{E}_i})$ do         ▷ verify authenticity
12:         $\mathsf{IntID}_{\mathcal{E}_j} \leftarrow \mathsf{IntID}^C_{\mathcal{E}_i}[idx]$
13:         $\mathsf{IntM}_{\mathcal{E}_j} \leftarrow \mathsf{IntM}^C_{\mathcal{E}_i}[idx]$
14:         if Verify$(\mathsf{IntID}_{\mathcal{E}_j})_{\mathcal{AK}} = Authentic$ then
15:             $\mathsf{IntM}.Join(\mathsf{IntM}_{\mathcal{E}_j})$
16:         else
17:             $\mathsf{IntM}[\mathcal{E}_j] \leftarrow$ “untrusted”
18:         end if
19:     end while
20:     $\mathsf{IntID} \leftarrow MAC(\mathsf{IntM}, \mathcal{N})_{\mathcal{AK}}$                ▷ authenticate
21:     $SendToParent(\mathsf{IntID}, \mathsf{IntM})$
22: end procedure
```

prevent leakage of $\mathcal{AK}$, the TSA encrypts it for each ECU to obtain $\mathsf{C}^{\mathcal{AK}}_{\mathcal{E}_i} = Enc(\mathcal{AK})_{\mathsf{SK}_{\mathcal{E}_i}}$ using the corresponding identity key $\mathsf{SK}_{\mathcal{E}_i}$. The encrypted keys $\mathsf{C}^{\mathcal{AK}}_{\mathcal{E}_1}, \mathsf{C}^{\mathcal{AK}}_{\mathcal{E}_2}, ..., \mathsf{C}^{\mathcal{AK}}_{\mathcal{E}_n}$ along with $\mathcal{N}$ are sent back to the vehicle, where $\mathcal{E}_C$ disseminates them to all $\mathcal{E}_i$.

Step 2 (Integrity Measurement). Upon receipt, $\mathcal{E}_i$ commences the process of measuring its software integrity by calling the procedure *Measure* (Algorithm 1) in a secure environment. The measurement of $\mathcal{E}_i$ produces an integrity identifier that we refer to as $\mathsf{IntID}_{\mathcal{E}_i}$. This value is the authenticated hash of selected software components, denoted as $\mathsf{SC}_{\mathcal{E}_i}$. These components encompass software on the functional layer, such as SOME/IP or ASOA services, ROS nodes, or custom binaries, often provided by third parties, as they are subject to regular extension and updates. To prevent $\mathcal{A}dv$ from penetrating lower levels of the software stack, including the Operating System (OS) or the bootloader, we also ensure the integrity of system files and the bootloader during *authenticated boot* (c.f., Section V-B).

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

Along with $\mathsf{IntID}_{\mathcal{E}_i}$, *Measure* produces an array IntM that contains the labels of the measured software and the corre-

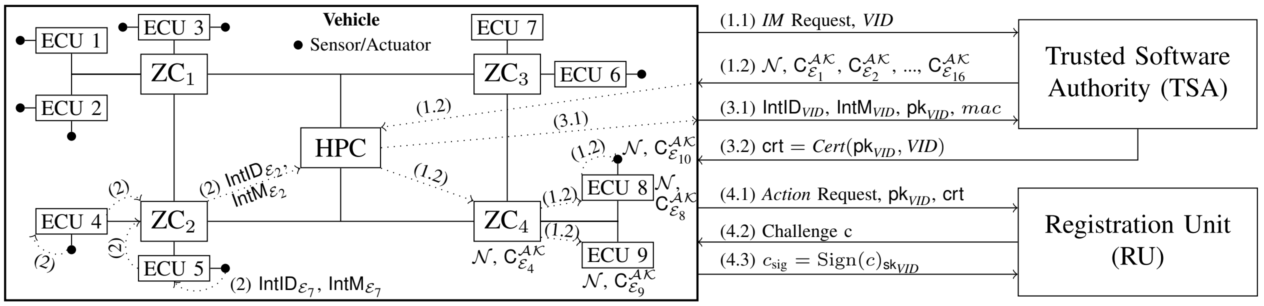
Vehicle
● Sensor/Actuator
ECU 1
ECU 2
ECU 3
ZC$_1$
ECU 7
ZC$_3$
ECU 6
HPC
(1.2)
(3.1)
(1.2)
(2) IntID$_{\mathcal{E}_2}$, IntM$_{\mathcal{E}_2}$
(2)
ECU 4
ZC$_2$
(2)
(2)
ECU 5
(2) IntID$_{\mathcal{E}_7}$, IntM$_{\mathcal{E}_7}$
ZC$_4$
$\mathcal{N}$, C$^{\mathcal{AK}}_{\mathcal{E}_4}$
(1.2) $\mathcal{N}$, C$^{\mathcal{AK}}_{\mathcal{E}_{10}}$
ECU 8
$\mathcal{N}$, C$^{\mathcal{AK}}_{\mathcal{E}_8}$
(1.2)
(1.2)
ECU 9
$\mathcal{N}$, C$^{\mathcal{AK}}_{\mathcal{E}_9}$
(1.1) *IM* Request, *VID*
(1.2) $\mathcal{N}$, C$^{\mathcal{AK}}_{\mathcal{E}_1}$, C$^{\mathcal{AK}}_{\mathcal{E}_2}$, ..., C$^{\mathcal{AK}}_{\mathcal{E}_{16}}$
(3.1) IntID$_{VID}$, IntM$_{VID}$, pk$_{VID}$, mac
(3.2) crt = *Cert*(pk$_{VID}$, *VID*)
Trusted Software Authority (TSA)
(4.1) *Action* Request, pk$_{VID}$, crt
(4.2) Challenge c
(4.3) $c_{\text{sig}} = \text{Sign}(c)_{\text{sk}_{VID}}$
Registration Unit (RU)

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

sponding hashes. IntM enables the TSA to identify software whose integrity check failed and factor this during validation. Given that an ECU may have several children, we utilize the array $\mathsf{IntM}^{C}_{\mathcal{E}_i}$ to store the IntM of $\mathcal{E}_i$'s children. Similarly, $\mathsf{IntID}^{C}_{\mathcal{E}_i}$ holds the integrity identifiers of $\mathcal{E}_i$'s children. In line 3, *Measure* first decrypts $\mathsf{C}^{\mathcal{AK}}_{\mathcal{E}_i}$ to receive the attestation key $\mathcal{AK}$. Then, starting in line 4, *Measure* performs integrity measurement by computing the hash of each software, which is added along with the software label to $\mathsf{IntM}_{\mathcal{E}_i}$ in line 7. Following this, *Measure* retrieves the results from the authenticated boot (line 9) to $\mathsf{IntM}[\mathcal{E}_i]$, allowing the TSA to verify the benign state of the OS and the bootloader. Finally, *Measure* verifies the authenticity of the measurements (line 11) carried out by $\mathcal{E}_i$'s children using the key $\mathcal{AK}$. This step is necessary to exclude manipulations of the measurements during their transmission within the vehicle. If the verification succeeds, the hashes and software labels of $\mathcal{E}_i$'s children are appended to $\mathsf{IntM}_{\mathcal{E}_i}$. Otherwise, $\mathcal{E}_i$ flags the software of the respective ECU as "untrusted", allowing the TSA to determine potential consequences during validation.

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

Step 3 (Validation). A trustworthy vehicle receives a certified digital identity that enables it to register with the RU. This certificate links the vehicle's identity *VID* to a public key pk_{VID}, which is generated by $\mathcal{E}_C$ alongside the corresponding secret key sk_{VID} every time the integrity measurement process terminates. Since $\mathcal{E}_C$ has no parent ECU within the vehicle, it sends IntID_{VID}, IntM_{VID}, and pk_{VID} to the TSA after terminating *Measure*. Furthermore, it transmits $mac = MAC(VID, \mathsf{pk}_{VID})_{\mathcal{AK}}$, allowing the TSA to verify the authentic origin of pk_{VID}. Upon reception, the TSA first verifies mac and IntID_{VID}. For that purpose, the TSA extracts the software labels from IntM, queries the original binaries from its database, and computes their hashes. If the verification succeeds immediately, the vehicle's software state is considered fully trusted. Otherwise, the TSA identifies the distrusted software and validates whether it must be considered critical or not. We propose implementing a validation process that considers the ECU on which the untrusted software runs and whether it has access to actuators and the potential to impact vehicle dynamics. Customization at or below the OS must be strictly prohibited. Therefore, the integrity of all boot stages must be ensured. That way, a customized application operating on the infotainment system may not pose a threat, while unverifiable software in a safety-critical zone must inevitably lead to countermeasures. If the validation judges the vehicle trustworthy, the TSA issues a certificate $\mathsf{crt} = Cert(\mathsf{pk}_{VID}, VID)$, which is eventually sent back to the vehicle. Otherwise, an error is returned, and consequently, $\mathsf{pk}_{\mathcal{E}_C}$ remains uncertified, rendering it impossible for the vehicle to register with the RU.

Step 4 (Registration). The registration of a vehicle with the RU entails the process of applying for a specific automotive action, which may vary depending on the mobility concept being employed. In either case, a secure communication channel is necessary to perform the requested action, which gives the possibility to exclude illegitimate vehicles. The registration is a challenge-response mechanism between a vehicle and the RU. Apart from the desired action, the request contains the vehicle's public key pk_{VID} and the previously obtained certificate crt. Once the request is received, the RU verifies the certificate and responds with a challenge, denoted as c. Upon reception, the requesting vehicle computes the signature $c_{\text{sig}} = \text{Sign}(c)_{\mathsf{sk}_{VID}}$ and sends it back to the RU. If the RU successfully verifies c_{sig}, the vehicle is considered trustworthy, and the requested action is permitted by establishing a secure channel between the RU. This is achieved through the use of the trusted public key pk_{VID}, which may be employed, for example, through a Diffie-Hellman key exchange. Otherwise, the request is rejected, and the vehicle is excluded from further actions. Hence, successful registration is contingent upon the vehicle having undergone an integrity measurement process prior to the request. Otherwise, the authenticity of pk_{VID} cannot be proven to the RU as the certificate crt would not have been issued.

Step 5 (Invalidation). To prevent $\mathcal{A}dv$ from using certificates representing an outdated software state, the TSA revokes the issued certificate when a vehicle is powered off. To achieve this, the central ECU implements a notification procedure *Notify* to inform the TSA when the vehicle is shut down. Similar to *Measure*, this procedure is executed in a secure environment.

A. Security Requirements

The genuine and secure execution of *Measure* and *Notify* is the fundamental assumption of our scheme. The problem, however, is that these functions are executed in a possibly compromised environment since our attacker model assumes the presence of $\mathcal{A}dv$ on the ECUs. Therefore, both procedures and the secret identity key $\mathsf{SK}_{\mathcal{E}_i}$ are part of the Trusted Computing Base (TCB), which is inaccessible to potential adversaries. This design ensures that even if $\mathcal{A}dv$ gains access to an ECU, it is impossible to tamper with or prevent *Measure* from executing as intended, i.e., it must not be interrupted by any other process. Technically, the TCB is protected through enclaves, which provide an isolated and secure environment for running trusted code, commonly referred to as *secure world*. As elaborated in [16], *Measure* and *Notify* must be kept in a write-protected memory area to preclude tampering. Similarly, the secret identity key $\mathsf{SK}_{\mathcal{E}_i}$ must be safeguarded against illegitimate access, which can be achieved through secure memory. Exclusive access to $\mathsf{SK}_{\mathcal{E}_i}$ can be enforced using an I/O Memory Management Unit (IOMMU).

B. Authenticated Boot

Beyond applications at the functional layer, it is essential to incorporate low-level software components into the integrity measurement process, particularly if ECUs are equipped with a full-stacked OS. $\mathcal{A}dv$ may compromise components such as the bootloader to gain control of the ECU. To address this, we employ authenticated boot, a technique to ensure that a computer system loads in an expected and trustworthy state. The boot process typically involves multiple stages that form a chain. The process usually begins with a code snippet from read-only memory that contains the logic to select the boot device where the first-stage bootloader (FSBL) is expected. The FSBL initializes basic hardware controllers and loads the second-stage bootloader (SSBL), for example, a U-Boot environment. The latter loads the Linux kernel space, which in turn loads the Linux user space where automotive applications run. During authenticated boot, the integrity of each stage of this boot chain is measured. The measurement results are stored in memory rather than verified immediately, as is the case with *secure boot*. We retrieve these results in line 9 of Algorithm 1 and integrate them into the integrity identifier as a single hash.

VI. Implementation & Evaluation

In this Section, we first describe our prototype implementation and then use it to evaluate the results of the proposed scheme.

A. Implementation

1) Physical Setup: Our physical setup is based on a recently presented cutting-edge zonal E/E architecture for safe and autonomous road vehicles [23]. The architecture consists of four interconnected zones that operate within an Ethernet network and is logically inspired by the structure of the human nervous system. It includes a *perception* unit, a *cerebrum*, a *brainstem*, and a *spinal cord*. While the perception processes data from sensors such as radar, lidar, and cameras, the spinal cord applies low-level commands to the vehicle's actuators. We have been involved [1] in implementing this E/E architecture in four automated prototype vehicles [24] with predominantly ARM-based ECUs, due to their inherent safety and security features. For instance, modern ARM processors support a dual-core lockstep mode and use the TrustZone technology for software isolation. Inside the vehicles, the brainstem is the central component capable of communicating with remote entities. It employs a Xilinx Ultrascale+ ZU3EG-1E, which features an ARM Cortex-R5 processor for real-time processing and an ARM Cortex-A53 processor for general-purpose computing. Non-resource-intensive tasks such as battery management, HMI, and door control are handled by off-the-shelf Raspberry Pis (RPis), which obviously do not meet automotive standards but are well-suited for demonstration purposes. In our setup, we recreated this zonal architecture in a best-effort approach using the original hardware where possible. That means we utilized the Ultrascale platform for the central ECU $\mathcal{E}_{\mathrm{C}}$ and RPis for the remaining controllers. To implement the proposed scheme, each ECU needs to know its parent and child nodes within the E/E architecture. Figure 3 illustrates the hierarchical arrangement of the ECUs schematically. Each zone ultimately

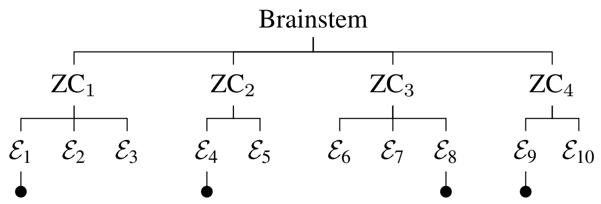

Fig. 3: The zonal E/E architecture replicated by our setup consists of four hierarchical layers: (1) the central brainstem as the root node, (2) the zone controllers, (3) the ECUs, and (4) sensors and actuators.

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

[1]https://www.unicaragil.de/en/

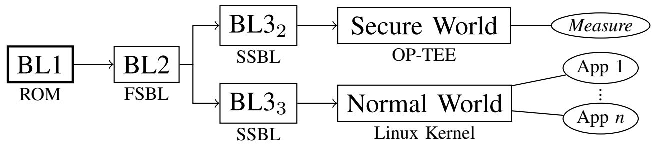

Fig. 4: The ARM boot chain uses a separate SSBL to load a secure world, in which we execute the *Measure* function.

2) OP-TEE: To ensure compliance with the security requirements outlined in Section V-A for the execution of *Measure* and *Notify*, we utilize the publicly available Open Portable Trusted Execution Environment (OP-TEE) [2], a framework leveraging the ARM TrustZone technology to create a *secure world* in which applications and data are isolated and protected from access by the *normal* world. This approach builds upon prior work [16] in which we proposed an attestation scheme for ECUs running a full-stack OS. OP-TEE uses the ARM Security Extensions to establish a Trusted Execution Environment (TEE) inside the secure world using two core building blocks: A Trusted Application (TA) is a signed binary that runs in the secure world and has access to shielded cryptographic features such as secure memory. In contrast, the Secure Monitor (SM) starts and stops TAs and manages communication channels between the secure and normal worlds. Note that while resources in the secure world may access the normal world, the reverse is impossible. In our implementation, we developed the *Measure* and *Notify* procedures in C++ as TAs. *Measure* uses the Linux Integrity Management Architecture (IMA) to perform the hashing of the targeted applications in SC. At the moment, we include *all* files in user space into SC to facilitate the creation of a policy for the IMA kernel module. The specification of SC may be adjusted to include or exclude specific files from processing. To allow ECUs to compute an aggravated integrity identifier, we include the addresses of the child and parent ECUs into each instance of *Measure*. We argue that hardcoding these addresses seems appropriate as the network topology remains static. The hashes produced by the IMA are securely stored and made available to *Measure* in the TEE for processing them as described in Algorithm 1. To guarantee the integrity of the OP-TEE environment, we integrate it into the authenticated boot process, as illustrated in Figure 4. For that purpose, we leverage the ARM boot chain, which provides dedicated stages for loading a TEE within the secure world. Specifically, the BL32 boot stage loads the SM and then the OP-TEE while the BL33 stage loads the Linux kernel in the normal world.

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

[2]https://www.op-tee.org/

B. Evaluation

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

1) Validation Overhead: We began by determining the boot time of each ECU without our scheme enabled to obtain a baseline. This involved loading a bare Linux OS without performing authenticated boot or computing integrity identifiers. We observed an average boot time of 17.2s on the RPis platforms and 15.3s on the central Xilinx Ultrascale+ system. Next, we built the OP-TEE environment, deployed it on our setup's ECUs, and initiated the *Measure* procedure in the secure world. During the integrity measurement process, we generated hashes of *all* files stored on the ECUs. Specifically, the RPi platform contains 130,578 system files, while the Xilinx Ultrascale+ board has 48,738 files. Note that the integrity of system files is expressed as a single hash, as described in line 9 of Algorithm 1. Hence, the TSA can recognize an untrusted OS but cannot identify the corrupted files, which is unnecessary as we always require a benign OS. In contrast, automotive applications in $\mathsf{SC}_{\mathcal{E}_i}$ are individually included in the integrity identifier, as these applications are typically added or updated and, thus, require precise validation. To simulate these applications, we created four dummy ASOA [4] services for each ECU. This number originates from the earlier mentioned prototype vehicles, which deploy 4.3 automotive applications on average on every ECU. As shown in Table I, our scheme executes in 21.3s on the RPi platform and 18.4s on the Ultrascale+ board. Thus, the integrity measurement adds an overhead of 4.1s and 3.1s, respectively, to the original boot process.

While the execution time on a single ECU is a first reasonable performance estimate, we still need to consider the time a given ECU needs to wait for its children to terminate. Recall that an integrity identifier includes the measurements of all hierarchically lower positioned ECUs, resulting in a dependency among them. For instance, the zone controller ZC_1

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

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

shown in Figure 3 can only proceed computing IntID_{ZC_1} and IntM_{ZC_1}, after the ECUs $\mathcal{E}_1$, $\mathcal{E}_2$, and $\mathcal{E}_3$ have provided their results. In our setup, it takes 21.9s from booting the system to the computation of the integrity identifier for the ECU on the highest hierarchical layer, i.e., the brainstem. Hence, we observe an additional latency of 600ms compared to the longest execution of *Measure* on a single device. This additional latency encompasses the network delay and the waiting time of the ECUs for their children to terminate.Ideally, the ECUs are arranged in a flat tree since the integrity measurement can occur fully parallel as only the root node needs to wait. In the worst case, the ECUs are arranged in a chain where each ECU has to wait for the adjoining one. We modified our setup to implement the latter case and observed that the computation of the vehicle's integrity identifier consumes 23.1s, corresponding to an overhead of 1.8s. We conclude that the hierarchical arrangement of the ECUs does impact the scheme execution time. However, this overhead is relatively low compared to the time necessary to boot and measure the individual files.

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

All in all, we have a total average overhead of 5.236s, which we consider acceptable since this time is added to the vehicle booting process. Most of the time is necessary for measuring system files. We can further reduce the scheme's overhead by adequately configuring the OS and minimizing its overall size.

2) Registration Overhead: At last, we aimed to determine whether the latency of the registration step is acceptable in a real scenario. To answer this question, we simulated an ecosystem in which *traffic entities* within a Range of Awareness (RoA) engage in continuous communication to optimize traffic and improve safety. For example, they might share sensor data to enhance situational awareness, especially in cases where obstacles or events are hidden from their direct perception. A traffic entity describes a communicating automotive object such as a vehicle, a roadside unit, or a traffic symbol. A vehicle seeking to join automated traffic must provide evidence of its trustworthiness to all entities in the RoA, i.e., registration is necessary. In other words, the RU consists of all traffic entities within a given RoA. We used a Matlab simulation framework [25] to examine the average time required for a vehicle to register with traffic entities in its RoA. This framework enabled us to simulate the data transmission based on Dedicated Short Range Communication (DSRC), which is based on the IEEE 802.11p standard. We assumed a moving vehicle; thus, new traffic entities appear in the RoA while others leave. Consequently, the registration step was continuously repeated while the vehicle was in motion. Table II summarizes the configuration parameters of our simulation. Our simulation considered both an urban and a highway scenario, both established by the ETSI [26]. The scenarios mainly differ in the density of obscuring objects, which affects the scattering behavior. The urban environment is densely populated with many roadside units, traffic signals, and vehicles. In contrast, our highway features three lanes with a primarily unobstructed line of sight. Vehicles existing on one end of the highway re-enter the same lane on the opposite one, requiring a new registration process as they are treated untrusted again. We modeled vehicle speeds in the urban and high environments using a Gaussian distribution with average speeds of 30 km/h and 50 km/h, and 90 km/h and 120 km/h, respectively. The standard deviation used for both environments was 12 km/h.

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

TABLE II: Technical parameters of our simulation

Our simulation encompassed multiple experiments investigating the registration overhead, i.e., the delay from a moving automobile to all traffic entities within its RoA. We considered densities of 50, 100, 150, and 200 traffic entities within a perimeter of one kilometer and RoAs of 50, 100, 200, and 300 meters. Note that the RoA generally has fewer traffic entities as it does not cover the whole scenario area. A single registration required the transmission of the certificate of trustworthiness crt (1048 KB), the public key pk_{VID} (1024 KB), and the action string (64 B) as described in the fourth step of our scheme. We also add a flat rate of 10ms to our results to account for the time required to verify crt and the challenge signed c, acknowledging that this value may vary on different systems.

Figure 5 visualizes the relationship between the registration overhead and the average number of traffic entities in all RoAs. Our results show that the delay grows with increasing traffic entities, although registrations are independent of each other and run in parallel; as expected more communication leads to more channel congestion and a larger delay. Additionally, on average, registration takes longer in an urban environment than on a highway, likely due to more obscuring objects, leading to a different scattering behavior. The registration delay ranges from 47 to 58.5ms which we found acceptable as the *minimal*-induced delay in such settings has been estimated to be approximately 40-50 ms [27].

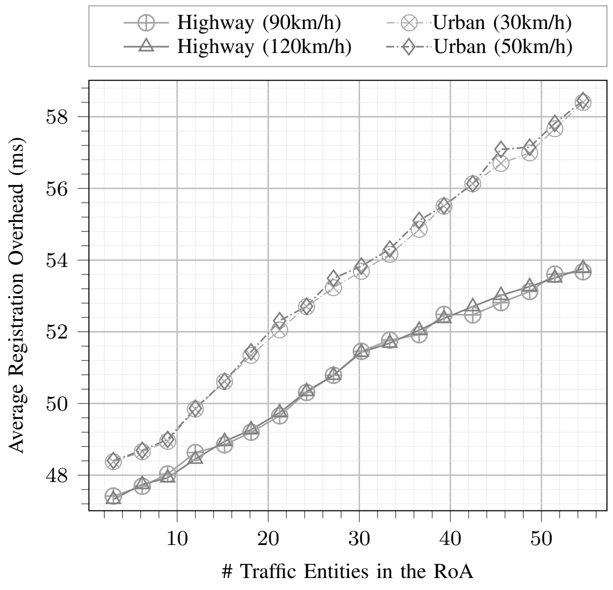

Fig. 5: The more traffic entities are in the RoA, the longer the registration takes. In an urban ETSI scenario, the registration generally lasts longer due to more obscuring objects than on a highway.

VII. Conclusion

In this work, we presented a novel scheme for determining and validating the software integrity of road vehicles. Unlike existing solutions, we do not rely on a trusted in-vehicle verification unit since such an approach does not scale for modern updatable systems. Instead, our scheme involves a remote component to validate a vehicle's integrity, allowing it to approve uncritical customizations or software from unknown sources. This component issues a certificate if it finds the vehicle's software trustworthy, enabling it to register with third parties requiring evidence of its trustworthiness. That way, other vehicles, roadside units, or authorities can ensure that vehicles do not pose a safety threat due to malicious software modifications. We employ hardware isolation techniques to securely compute an integrity identifier on potentially compromised systems in a cascading manner, leveraging the hierarchical structure of modern E/E architectures. We implemented the scheme on a prototype setup based on a recently presented zonal E/E architecture, evaluated its performance, and simulated the registration step in an urban and highway ETSI scenario. Our experimental results show that the proposed scheme currently incurs a total overhead of 5.236s to the vehicle's starting procedure, while the average registration delay ranges between 47-58.5ms. As part of our future work, we plan to focus on runtime attacks since, currently, we are unable to detect malicious system changes after an integrity measurement has been carried out on a running vehicle.

Acknowledgement

This work has been accomplished within the projects "AUTOtech.agil" (FKZ01IS22088X) and "6G-RIC" (16KISK034). We acknowledge the financial support for the projects by the Federal Ministry of Education and Research of Germany (BMBF).

References

[1] "CAN Specification," Robert Bosch GmbH, 1991, version 2.0.

[2] *SOME/IP Protocol Specification*, AUTOSAR, 11 2016, 1.0.

[3] "AUTOSAR Adaptive," https://www.autosar.org/standards/adaptive-platform, accessed: 2023-03-06.

[4] A. Kampmann, B. Alrifaee *et al.*, "A Dynamic Service-Oriented Software Architecture for Highly Automated Vehicles," in *2019 IEEE Intelligent Transportation Systems Conference*, 2019, pp. 2101–2108.

[5] S. K. Khan, N. Shiwakoti *et al.*, "Cyber-attacks in the next-generation cars, mitigation techniques, anticipated readiness and future directions," *Accident Analysis & Prevention*, 2020.

[6] C. Miller and C. Valasek, "Remote exploitation of an unaltered passenger vehicle," *Black Hat USA*, vol. 2015, p. 91, 2015.

[7] Z. Cai, A. Wang *et al.*, "0-days & mitigations: roadways to exploit and secure connected bmw cars," *Black Hat USA*, vol. 2019, p. 39, 2019.

[8] M. Gierl, R. Kriesten, and E. Sax, "Security Assessment Prospects as Part of Vehicle Regulations," in *Computer Safety, Reliability, and Security. SAFECOMP 2022 Workshops*, M. Trapp, E. Schoitsch *et al.*, Eds. Cham: Springer International Publishing, 2022, pp. 97–109.

[9] "Road vehicles - Cybersecurity engineering," International Organization for Standardization, Geneva, CH, Standard, Mar. 2021.

[10] "UN Regulation No 156 – Uniform provisions concerning the approval of vehicles with regards to software update and software updates management system [2021/388]," pp. 60–74, Mar 2021.

[11] G. Coker, J. Guttman *et al.*, "Principles of remote attestation," *International Journal of Information Security*, vol. 10, pp. 63–81, 2011.

[12] C. Shepherd, K. Markantonakis, and G.-A. Jaloyan, "Lira-v: Lightweight remote attestation for constrained risc-v devices," in *2021 IEEE Security and Privacy Workshops (SPW)*, 2021, pp. 221–227.

[13] I. D. O. Nunes, K. Eldefrawy *et al.*, "Vrased: A verified hardware/software co-design for remote attestation." in *USENIX Security Symposium*, 2019, pp. 1429–1446.

[14] J. Erickson, S. Chen *et al.*, "CommPact: Evaluating the Feasibility of Autonomous Vehicle Contracts," in *2018 IEEE Vehicular Networking Conference (VNC)*, 2018, pp. 1–8.

[15] D. Püllen, N. A. Anagnostopoulos *et al.*, "Poster: Hierarchical Integrity Checking in Heterogeneous Vehicular Networks," in *2018 IEEE Vehicular Networking Conference*, 2018.

[16] F. Kohnhäuser, D. Püllen, and S. 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.

[17] *Specification of Secure Onboard Communication*, AUTOSAR, 2019, r19-11.

[18] D. Zelle, T. Lauser *et al.*, "Analyzing and Securing SOME/IP Automotive Services with Formal and Practical Methods." Association for Computing Machinery.

[19] M. Iorio, M. Reineri *et al.*, "Securing SOME/IP for In-Vehicle Service Protection," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 11, pp. 13 450–13 466, 2020.

[20] "OMG DDS Security," https://www.omg.org/spec/DDS-SECURITY/1.1/PDF, 2018, version 1.1.

[21] J. Feiler, S. Hoffmann, and D. F. Diermeyer, "Concept of a control center for an automated vehicle fleet," in *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*, 2020, pp. 1–6.

[22] "Entwurf eines Gesetzes zur Änderung des Straßenverkehrsgesetzes und des Pflichtversicherungsgesetzes – Gesetz zum autonomen Fahren," https://dserver.bundestag.de/btd/19/274/1927439.pdf, accessed: 2023-02-03.

[23] D. Zaheri, D. Niedballa *et al.*, "Practical implementation of a zonal e/e architecture for autonomous vehicles in unicaragil," *ATZelectronics worldwide*, vol. 18, no. 1, pp. 8–12, 2023.

[24] T. Woopen *et al.*, "UNICARagil–Disruptive Modular Architectures for Agile, Automated Vehicle Concepts," in *27th Aachen Colloquium Automobile and Engine Technology 2018*, Oct 2018, pp. 663–694.

[25] V. Todisco, S. Bartoletti *et al.*, "Performance Analysis of Sidelink 5G-V2X Mode 2 Through an Open-Source Simulator," vol. 9, pp. 145 648–145 661, 2021.

[26] "Intelligent Transport Systems (ITS); Access Layer; Part 1: Channel Models for the 5,9 GHz frequency band," European Telecommunications Standards Institute, Technical Report, 2019.

[27] A. Baiocchi, I. Turcanu *et al.*, "Age of information in ieee 802.11p," in *2021 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, 2021.