2019 IEEE Security and Privacy Workshops (SPW)

Ensuring the Safe and Secure Operation of Electronic Control Units in Road Vehicles

Florian Kohnhäuser*, Dominik Püllen†, Stefan Katzenbeisser‡
Security Engineering Group, Computer Science Department, TU Darmstadt, Darmstadt, Hessen, Germany
Emails:{kohnhaeuser*, puellen†, katzenbeisser§}@seceng.informatik.tu-darmstadt.de

***Abstract*—With the increasing connectivity and complexity of road vehicles, security heavily impacts the safety of vehicles. In fact, researchers demonstrated that the lack of security in vehicles can lead to dangerous and even life-threatening situations. A threat that has been insufficiently addressed in existing vehicular security solutions are software attacks, in which the adversary compromises the software of Electronic Control Units (ECUs). A promising technique to defend against software attacks is remote attestation, as it enables to detect compromised devices. This paper presents a novel attestation scheme that ensures the software integrity of ECUs to warrant the vehicle's safety. In our scheme, a trusted master ECU verifies the integrity of all safety-critical ECUs and refuses to start the engine in case an untrustworthy, and hence, unsafe state is detected. As modern vehicles are highly heterogeneous system of systems, we propose two different attestation techniques that enable the attestation of simple ECUs, such as basic sensors or actuators, as well as advanced, more complex ECUs like sensor fusion systems. We implement our attestation scheme on an exemplary automotive network that incorporates CAN and Ethernet, and show that our solution imposes an imperceptible overhead for passengers.**

***Index Terms*—safety, security, remote attestation, road vehicles**

I. Introduction

In the past decades, vehicles converted from mere mechanical devices to complex systems with dozens of interconnected embedded computers, called Electronic Control Units (ECUs). Nowadays, ECUs are accessible through various communication protocols and interfaces, of which some are even wireless, e.g., Bluetooth, WiFi, and LTE. This evolution drastically increased the attack surface of vehicles and rendered established techniques to ensure the safety of cars, like redundancy, reliability, and determinism, insufficient. In fact, recent attacks [16] and security evaluations [5], [9] have demonstrated that ECUs can be manipulated to perform malicious actions which can lead to life-threatening accidents. Consequently, the safety of modern vehicles can only be guaranteed by establishing an appropriate level of security.

To secure vehicles, much effort has gone into the research [18], [20] and standardization [4] of authentication protocols. These protocols protect against network attacks, in which the adversary attempts to forge or manipulate messages in the automotive network. Although authentication protocols are essential for security, they cannot protect against all attacks. In particular, they are unable to defend against software attacks, in which the adversary compromises the software of ECUs to perform malicious actions. For instance, the firmware of brake actuators may be manipulated to refuse applying the actual brakes. Authentication protocols are unable to prevent this attack. A key technology to defend against software attacks is remote attestation. It is typically realized as a challenge-response protocol between a verifier and at least one prover. By executing the protocol, the verifier determines whether the prover is in a trustworthy software state (or not).

The potential of remote attestation to protect vehicles against software attacks has already been recognized in existing works. However, existing solutions suffer from limited applicability, as they either require ECUs to implement a Trusted Platform Module (TPM) [14], [19] or the SANCUS research architecture [25]. As of yet, ECUs are not equipped with TPMs and SANCUS is commercially unavailable. Furthermore, existing works do not focus on safety [14], [19], [25] or lack an implementation and evaluation of the attestation [25].

Contributions. In this work, we present a novel solution to ensure the safe and secure operation of ECUs in road vehicles. In short, we make the following contributions:

- Automotive Attestation Scheme: We present the first attestation scheme that aims to ensure the safety of road vehicles. In our scheme, a trusted master ECU verifies the software integrity of all safety-critical ECUs in the vehicle, each time passengers unlock the vehicle and open the door(s). Only after all safety-critical ECUs are successfully verified, the master ECU mechanically allows the engine to launch.
- Applicability to Commodity ECUs: To address the heterogeneous character of ECUs in road vehicles, our scheme comprises two attestation techniques. The first technique is suited for simple ECUs that possess only a small and modest system architecture, such as basic sensors and actuators. The second technique targets advanced ECUs that feature ARM application processors, and hence are suited for more complex tasks, e.g., sensor fusion. We show that both techniques are applicable to a broad range of existing commodity ECUs which are deployed in road vehicles. Thus, our scheme entails only minimal deployment costs.
- Evaluation: We implement our scheme on commodity devices, which we connect in an automotive CAN and Ethernet network. Measurements conducted in our setup demonstrate that 100 ECUs can be verified in less than 1.4 seconds. Note that typical vehicles contain 70-100 ECUs in total, so that the amount of safety-critical ECUs can be expected to be less than 100. Based on our measurement results, we argue that passengers are unable to notice any overhead to the startup time of vehicles.

Outline. § II summarizes related work. § III describes our system model. § IV presents our attestation scheme. § V shows our implementation and evaluation. § VI concludes this work.

DOI 10.1109/SPW.2019.00032

II. Related Work

Remote Attestation. Remote attestation is a technique that allows a third party, the *verifier*, to ensure the integrity of a remote device, the *prover*. During attestation, the verifier challenges the prover and receives a response that indicates whether the prover is in a trustworthy system state (or not). *Hardware-based* attestation techniques [13] rely on secure hardware that is built-in the prover device, such as the TPM, ARM TrustZone, or Intel SGX. For low-end embedded devices, such hardware has shown to be too complex and expensive. This is why recent techniques aim at an attestation with minimal secure hardware [10], [17]. In contrast, *software-based* attestation is hardware-independent and thus suited for legacy devices that do not provide secure hardware. However, software-based techniques rely on strong assumptions that are hard to achieve in practice [3], such as an optimal implementation and execution of the protocol, precise time measurements, and an adversary who is passive during attestation.

Attestation of ECUs in Vehicles. Oguma et al. [14], [19] proposed an attestation scheme that enables a dedicated master ECU to verify the integrity of other ECUs within the vehicle. To ensure a secure attestation, the master and all ordinary ECUs are equipped with a TPM 1.2. Yet, TPMs are considered too complex and costly to be deployed in embedded systems. Hence, the proposed attestation scheme suffers from limited applicability in actual vehicles, where most, if not all, ECUs are embedded systems. We acknowledge that the TPM 2.0 Automotive Thin Profile was recently announced. Yet, because these new TPMs are also separate microcontrollers that need to be integrated into ECUs, they increase the ECUs' size, complexity, and cost, just like the TPM 1.2 [26]. Moreover, these TPMs are only commercially available since October 2018 and are not deployed in current commodity ECUs.

VulCAN [25] is a message authentication, software isolation, and attestation scheme, which is able to dynamically attest the integrity of ECUs across an untrusted vehicle network. However, VulCAN builds on the assumption that all attestable ECUs are equipped with SANCUS [17], which is a research security architecture that is commercially unavailable. Furthermore, VulCAN's attestation concept was not implemented and evaluated, so that its practicality has not been proven.

Attestation of Commodity Embedded Devices. Recent works have shown that commodity low-end embedded devices can already be attested in case they provide read-only boot code and a simple memory protection mechanism [6], [11], [12], [23]. However, none of the works investigated whether the required hardware security features can be found in ECUs. Moreover, the proposed solutions were not implemented and evaluated in automotive networks. Hence, their practicality in the automotive context is unclear.

Other works approach remote attestation on more powerful commodity embedded devices that additionally feature the ARM TrustZone technology. fTPM [21] and Komodo [8] show how a TPM, respectively Intel SGX, can be implemented in software using the TrustZone. Shepherd et al. [24] present a protocol that establishes secure channels between devices, based on a mutual attestation of applications that run in the TrustZone. Nevertheless, fTPM [21] and Komodo [8] only provide a first step towards attestation, as they lack a concept to measure and verify the integrity, and [24] only enables the attestation of a single program running in the TrustZone.

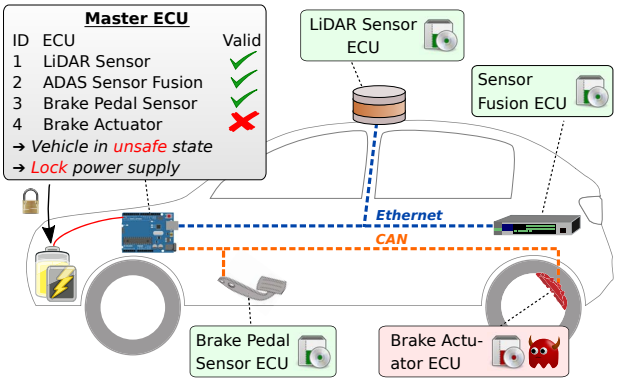

Fig. 1: Illustration of our system architecture in an electric car.

III. Preliminaries

System Model. As depicted in Figure 1, we consider a vehicle that consists of various interconnected ECUs. The *Master ECU* has a special role, because it verifies the software integrity of other ECUs in the vehicle using our proposed attestation scheme. The master is directly connected to the ignition switch in gasoline-powered vehicles or to the power supply in all-electric vehicles. In case the software of at least one safety-critical ECU cannot be verified, the master refuses to start the vehicle. The master is assumed to be trustworthy, as opposed to all other ECUs within the vehicle, which may be compromised by an adversary, as described below.

To be verifiable by the master, ECUs must provide certain secure hardware features, which are stated in § IV-B. We focus on two types of ECUs. First, we consider *simple ECUs* that perform basic sensor and/or actuator tasks, such as sensing the brake pedal or applying the actual brakes. Simple ECUs are typically low-end to mid-range devices with a few MB of storage and a CPU that operates at most at a few hundred MHz. Second, we regard more *advanced ECUs*, which are mid-range to high-end embedded devices that run a full-featured Operating System (OS) including various applications. Advanced ECUs are used in more complex tasks, such as sensor fusion or object detection in Advanced Driver Assistance Systems (ADASs). Although our solution is suited to attest all ECUs in the vehicle, this may not be necessary from a safety perspective. For instance, assuming that all in-vehicle communication is authenticated, a window lifter has a negligible impact on safety compared to the brakes.

Adversary Model. We assume an adversary $\mathcal{A}dv$ who has full control over the communication medium (Dolev-Yao model). Thus, $\mathcal{A}dv$ can eavesdrop, modify, delete, or insert any message between all ECUs in the network. We further assume that $\mathcal{A}dv$ can compromise the software of all ECUs, except for the trusted master ECU. On a compromised ECU, $\mathcal{A}dv$

has complete control over the execution state and can read all readable storage and write to all writable storage. However, we assume that $\mathcal{A}dv$ is unable to bypass the hardware protection provided by an ECU. Thus, $\mathcal{A}dv$ cannot manipulate data or code that is stored or executed inside hardware-protected memory. In addition, we do not consider Denial of Service (DoS) attacks on individual ECUs, as they cannot be prevented against an adversary who has full control over the network. We refer to our attestation scheme as secure, if $\mathcal{A}dv$ is unable to forge a valid system state for a software-compromised ECU.

IV. Attestation Scheme for Road Vehicles

In this section, we first describe the basic functioning of our attestation scheme (§ IV-A) and then explain its technical implementation details on simple and advanced ECUs (§ IV-B).

A. Overview

The center of our attestation scheme constitutes the master ECU. At each time the vehicle is unlocked and then opened, the master verifies the software integrity of a predefined set of safety-critical ECUs by executing our proposed attestation protocol. The set of safety-critical ECUs is fixed at the roll-out of the vehicle by the manufacturer, but can be changed later on, e.g., in case ECUs need to be replaced. The manufacturer deploys each safety-critical ECU with two different nonces, $\mathcal{N}_B$ and $\mathcal{N}$, a unique identifier ID, and a unique attestation key AK that is only known to the ECU and the master. Furthermore, the master and all safety-critical ECUs are supplied with the protocol code and data, as detailed in § IV-B. The protocol is illustrated in Figure 2 and described in the following.

The attestation takes place in three steps that are sequentially executed when the vehicle is unlocked and opened:

Step (1). All safety-critical ECUs boot and measure their local software integrity. To this end, ECUs compute a hash value over their installed software and store it in IM, the integrity measurement variable. Afterwards, each ECU derives a so-called response key RK. ECUs generate RK by computing an Hash-based Message Authentication Code (HMAC) over the nonce $\mathcal{N}_B$ and the measurements IM with the attestation key AK. In step (2), RK is used to answer the attestation challenge from the master. If an ECU is in a compromised software state, IM differs from the ECU's known good integrity measurement, so that RK does not match the ECU's expected response key. In step (3), this is eventually detected by the master ECU when verifying the attestation response.

For a secure attestation, it is crucial that an adversary $\mathcal{A}dv$ is unable to obtain access to the attestation key AK of ECUs, and cannot tamper with the computation of IM and RK. The details to ensure this depend on the ECU type and are subject to the following subsection (§ IV-B).

Step (2). The master ECU generates a random nonce $\mathcal{N}$ and broadcasts it to all safety-critical ECUs. The nonce $\mathcal{N}$ functions as a challenge and prevents $\mathcal{A}dv$ from performing replay attacks with recorded attestation responses.

Upon the reception of $\mathcal{N}$, ECUs overwrite $\mathcal{N}_B$ with $\mathcal{N}$. Since RK is dependent on $\mathcal{N}_B$, this causes RK to change in each attestation run. As discussed later on, this enables our scheme to recover from so-called runtime attacks, in which $\mathcal{A}dv$ compromises the software of ECUs after their attestation. Afterwards, ECUs generate their attestation response, which consists of two parts: (i) the identity ID of the ECU, and (ii) an HMAC σ that is computed over $\mathcal{N}$ and ID with the response key RK from step (1). Finally, ECUs send their attestation response, i.e., ID and σ, to the master ECU.

Step (3). The master ECU receives all responses and verifies them as follows. For each safety-critical ECU, the master stores the known good integrity measurement IM' as well as the secret attestation key AK'. Based on the ID contained in a received response, the master retrieves the known good integrity measurement IM' and the attestation key AK' of an ECU. Using AK' and IM' as well as the previous nonce $\mathcal{N}_B$, the master then computes the expected response key RK'. Next, the master generates an HMAC over N and ID with the computed RK', and compares it with σ from the attestation response. If both values match, the attestation response of the particular ECU is considered valid.

Only if all safety-critical ECUs replied with a valid response, the master ECU regards the vehicle to be in a safe state. Only then, the master allows the vehicle to launch the engine by releasing the lock for the ignition switch and/or power supply. Hence, compromised ECUs can only prevent the start of the vehicle, but are unable to jeopardize its safety.

B. Technical Details

Security Requirements. For a secure attestation, it is crucial that an adversary $\mathcal{A}dv$ (i) is unable to access AK, and (ii) cannot manipulate the computation of the software integrity measurements IM and the response key RK. If (i) and (ii) are ensured, $\mathcal{A}dv$ cannot compute RK. In addition, RK will reflect the software integrity of the ECU. This means that RK only matches the expected response key RK', which is used by the master to verify the attestation response, in case the respective ECU is in a trustworthy software state. Since $\mathcal{A}dv$ can only bypass our attestation scheme by violating (i) or (ii), all hardware and software that ensures (i) and (ii) is our Trusted Computing Base (TCB).

More specifically, our TCB software consists of all code that is executed in step (1), in which AK is accessed and IM and RK are computed. Common for hardware-based attestation schemes, we require the TCB software to be supported by secure hardware that must provide the following properties [10]:

(1) *Secure Storage*: The attestation key AK must exclusively be accessible to the TCB code.
(2) *Immutability*: The TCB code and data must be stored in a write-protected memory region.
(3) *Uninterruptability*: Once the TCB code gets executed, its execution cannot be interrupted by other code.

In the following, we discuss the implementation of these properties and the TCB code on simple and advanced ECUs.

Implementation on Simple ECUs. Simple ECUs are deployed with a bootloader that is stored in Read-Only Memory

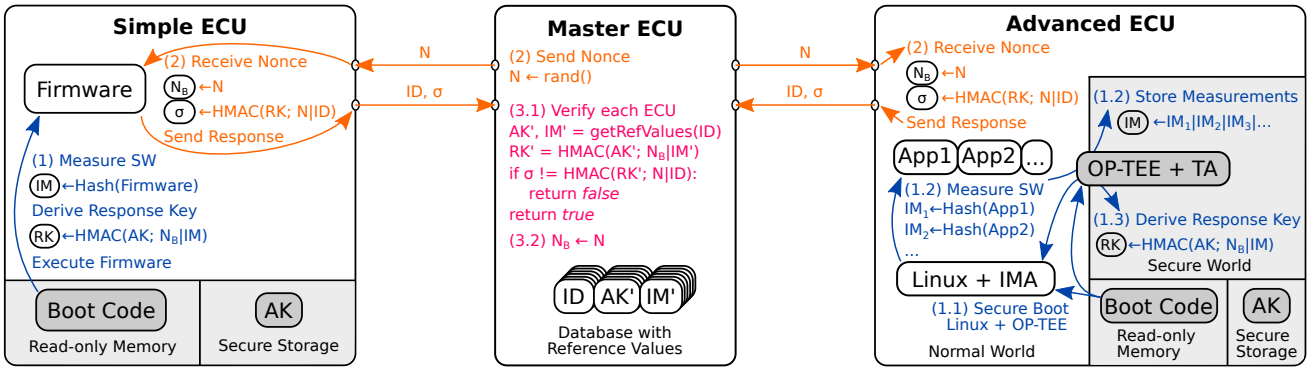

Fig. 2: Attestation of simple and advanced ECUs by the master ECU. Code and data that is protected by secure hardware is shown in gray.

(ROM) and contains the TCB code. Executing the bootloader, the TCB code first computes IM by hashing predefined memory regions that contain the firmware of the ECU. To increase flexibility, the memory regions to be measured can be read out from a certificate, as described in [12]. Next, TCB code accesses AK and generates the response key RK. After generating RK, TCB code ensures that AK is hidden, which may involve purging intermediate secrets from memory and locking AK against further access by software. Finally, the bootloader executes the measured firmware.

Storing the TCB code in the write-protected bootloader ensures that it is immutable and immediately executed when the ECU starts. By temporarily disabling all interrupts, the bootloader code also ensures its own uninterruptability until the TCB code finishes execution. Consequently, any malicious code can only be executed after step (1). The implementation of the secure storage depends on the respective hardware of an ECU. Existing works have demonstrated concrete implementations based on a simple Memory Protection Unit (MPU) [11], an emulated MPU [6], an SRAM PUF [12], [23], or EEPROM that can be hidden [12], [23]. In § V-A, we describe the implementation of all properties on an exemplary device.

In practice, many simple ECUs provide the required secure hardware properties, as they are based on microcontrollers that have shown to exhibit those properties [6], [11], [12], [23]. Furthermore, since several years, all major manufacturers offer ECUs[1] that fulfill the HIS-SHE [7] and/or EVITA [26] standard. Both standards provide a common specification for secure hardware in road vehicles. They specify that ECUs must offer hardware support for immutable and uninterruptible code to implement the secure boot functionality [2], and thus fulfill property (2) and (3). Based on (2) and (3), recent work [6] has shown that a secure storage can be emulated to also fulfill property (1). Therefore, all ECUs that implement the HIS-SHE and EVITA standard also implicitly fulfill our secure hardware requirements. Note that actual ECUs often offer a real MPU, whose use we prefer over an emulated MPU (e.g., all ECUs mentioned in the footnote[1] provide an MPU).

Implementation on Advanced ECUs. Advanced ECUs utilize the same measures as simple ECUs to implement the necessary properties. Hence, they also store their boot code and boot data in ROM. Yet, to achieve increased functionality, we require advanced ECUs to also feature the ARM TrustZone technology [1], which offers a privileged second execution environment, called secure world. The secure world is isolated by hardware from the normal world, in which the standard OS and applications are running. Since the ARM Cortex-A15 from 2010, all ARM application processors (Cortex-A) feature the TrustZone technology. Moreover, the new ARMv8-M architecture brings TrustZone to microcontrollers, namely Cortex-M23/M33/M35P processors. Additionally, we require the secure storage of AK to be only accessible from the secure world, which is a common requirement for TrustZone-based security services [8], [21]. To implement the secure storage, manufacturers offer different solutions, such as hardware fuses, TrustZone aware memory controllers, and/or IOMMU [8].

The additional secure hardware features are necessary, because advanced ECUs must rely on a different attestation technique than simple ECUs due to their software complexity. Nowadays, a full-featured OS contains thousands of constantly changing files. Predefining which memory regions must be measured to detect malware, as required for simple ECUs, would hence be an impossible task on advanced ECUs. Therefore, we instead present a flexible technique, where any executable code is measured on demand at load-time, before it is executed. Using the additional security features of advanced ECUs, our technique ensures that the measurement code is unmodified, and executed potentially malicious software is unable to tamper with already performed measurements.

More precisely, advanced ECUs are deployed with a bootloader that performs a secure boot [2] of two software systems: (i) a Linux-based OS running in the normal world, and (ii) a software called OP-TEE [15] running in the TrustZone secure world. The secure boot technology ensures that the particular software comes from a trusted party, like the manufacturer, as opposed to an adversary. For this purpose, the software is measured and the measurement is compared with a reference measurement from a stored certificate. In case actual and reference measurement differ or the signature verification of the certificate fails, the software is not executed.

[1]E.g., AURIX from Infineon; MPC564xB/C, MPC5746M, MPC574xB-C-G from NXP; SPC564B/EC, SPC56ECx from ST; RH850/P1x-C from Renesas.

All (automotive) applications running on the advanced ECU are managed by the Linux-based OS, which is deployed with an enabled Integrity Measurement Architecture (IMA) [22]. IMA is part of recent Linux kernels and offers the capability to measure binary files before their execution, including libraries and configuration files. Each measurement is stored in the IMA measurement list and contains, among others, the filename, filepath, and hash value computed over the file.

The second securely booted software, OP-TEE, manages all applications that are running in the TrustZone secure world. In particular, a special Trusted Application (TA) is deployed in the secure world of advanced ECUs. The TA provides an API that enables the IMA to pass performed measurements to the TA. Passed measurements are concatenated in IM ($\mathsf{IM} = \mathsf{IM}_1|\mathsf{IM}_2|...$) and stored protected in the secure world. In addition, the TA offers a second API to compute the response key RK. For this purpose, the TA takes a nonce $\mathcal{N}_B$ and returns an HMAC RK that is computed with the attestation key AK over $\mathcal{N}_B$ and the concatenated measurements IM. Note that the computation of RK can also take place on demand in step (2). This is possible, as the computation is handled in the secure world, which prevents potentially malicious code running in the normal world from tampering with the computation.

Runtime Attacks. We acknowledge that *Adv* may also perform runtime attacks, in which *Adv* compromises the software of an ECU after it was measured. Such attacks allow *Adv* to subvert the security of our scheme and are a well-known limitation of all load-time attestation protocols, e.g., also affect the widely deployed TPM. Nevertheless, the boot nonce $\mathcal{N}_B$ prevents any persistent attacks, since it causes RK to change in each attestation run. Hence, malware that is installed by *Adv* during a runtime attack is detected at the next ECU restart.

V. Evaluation

In this section, we describe our implementation setup (§ V-A) and then show and evaluate our measurements (§ V-B).

A. Implementation

We connected exemplary simple and advanced ECUs in a CAN bus and via Ethernet. As a target platform for simple ECUs, we used 5 Olimex ESP32-EVB development boards, which feature 4 MB flash memory, a 240 MHz dualcore 32-bit microprocessor, 100 MBit Ethernet, and a CAN communication module. To implement the required security properties (§ IV-B), we locked the bootloader of each ESP32-EVB with the "one-time flash" option, such that it cannot be modified. Furthermore, our deployed bootloader configures the MPU to enable only the bootloader code access to the attestation key AK. For advanced ECUs and the master ECU, we employed 6 Raspberry Pi 3 Model B+, which feature 1 GB RAM, a 1.4 GHz quadcore ARM Cortex-A53, and Gigabit Ethernet. In addition, we extended each RPI3B+ with a SK Pang PiCAN2 module to enable CAN communication. Unfortunately, we were unable to implement all required security properties on the RPI3B+ (§ IV-B), as the RPI3B+ insufficiently protects TrustZone secure world memory and lacks a secure storage.

Operation	ESP32-EVB	RPI3B+
Secure Boot Overhead	587.38 ms	508.29 ms
SHA-256(32 B Flash)	0.04 ms	0.06 ms
SHA-256(512 KB Flash)	126.71 ms	67.30 ms
SHA-256(2 MB Flash)	-	281.65 ms
HMAC-SHA-256(32 B RAM)	0.10 ms	0.08 ms
HMAC-SHA-256(64 B RAM)	0.12 ms	0.09 ms
HMAC-SHA-256(1 KB RAM)	0.37 ms	0.32 ms
Ethernet Round Trip Time	0.75 ms	0.44 ms
Ethernet Throughput	49.08 MBit/s	95.70 MBit/s
CAN Round Trip Time	0.17 ms	0.85 ms
CAN Throughput	1.27 MBit/s	1.22 MBit/s

TABLE I: Averaged measurements on the ESP32-EVB and RPI3B+.

We further acknowledge that our selected hardware lacks certain safety features that can be found in typical automotive hardware, such as a lockstep mode, ECC memory, and a large operating temperature range. However, the implementation of a TrustZone-aware memory controller, secure storage, and automotive safety features only entail a negligible runtime overhead and thus have an insignificant effect on our performance measurements.

B. Runtime Measurements

Single-Device Measurements. We investigated the runtime overhead to verify a single device. Table I outlines our averaged runtime measurements for the main building blocks of our attestation scheme on the ESP32-EVB and the RPI3B+. As shown, the main overhead on both devices comes from the secure boot, which ensures the immutability and uninterruptability of the boot code (§ IV-B). The secure boot takes with 587 ms on ESP32-EVB and 508 ms on RPI3B+ much more time than other operations, as it involves multiple computationally expensive signature verifications. Note that on other devices, a secure boot may not be required to implement the secure hardware properties (§ IV-B).

Further overhead is imposed by cryptographic operations, which we implemented with the mbed TLS library. During attestation, both devices compute a SHA-256 hash value over all software to be executed. Whereas the ESP32-EVB hashes a 512 KB firmware binary, which takes 127 ms, the RPI3B+ measures 29 files that sum up to a total of 2.5 MB, which takes 326 ms. Moreover, both devices compute two HMACs, which consumes 0.24 ms runtime on the ESP32-EVB and 0.41 ms on the RPI3B+. The RPI3B+ requires more time, because it computes an HMAC over 29 software integrity measurements, instead of a single measurement. Furthermore, the master ECU, an RPI3B+, requires 0.18 ms to recompute the HMACs and verify the attestation response of an ESP32-EVB, and 0.44 ms to verify responses of an RPI3B+.

Additionally, the network communication also entails overhead, as a 16 Byte challenge and 33 Byte response needs to be transmitted during attestation. Interestingly, the performance

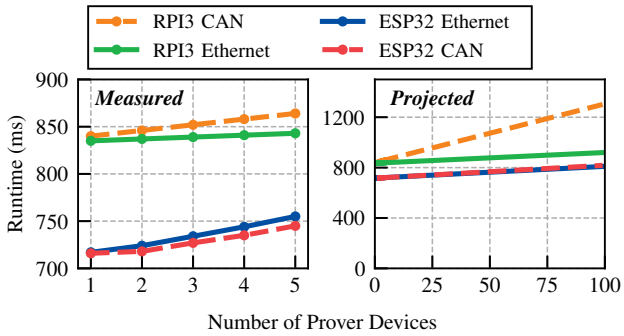

Fig. 3: Runtime overhead of verifying multiple ECUs.

of the network interfaces is quite different on the ESP32-EVB and RPI3B+. On the ESP32-EVB, CAN is faster than Ethernet and requires only 1.19 ms to transmit both challenge and response. By contrast, on the RPI3B+, CAN is much slower than Ethernet and entails an overhead of 5.95 ms.

Altogether, the runtime overhead to perform the attestation of a single device depends on the device type and network interface. It varies between at least 715 ms (on ESP32-EVB over CAN) and at most 844 ms (on RPI3B+ over CAN).

Multi-Device Measurements. Figure 3 depicts the runtime overhead to attest a varying amount of ECUs over CAN and Ethernet. As shown, the fastest attestation is possible with the ESP32-EVB over CAN, while the slowest is the RPI3B+ over CAN. Compared with the overhead to verify a single device, the runtime to attest many devices only slightly increases with a raising number of devices in the network. This demonstrates the good scalability of our attestation scheme. In fact, our projections to large networks show that the attestation of up to 100 devices takes even in the worst-case, being a CAN network with only RPI3B+ devices, less than 1.4 seconds. Because a typical passenger requires more than 1.4 seconds to get in the vehicle and initiate a start of the engine, our scheme entails no perceptible delays for passengers. Moreover, road vehicles typically contain less than 100 safety-critical ECUs.

VI. Conclusion & Future Work

In this work, we presented a novel automotive attestation scheme. In our scheme, a trusted master ECU verifies the software integrity of all safety-critical ECUs in the vehicle, each time the vehicle is unlocked and opened. Only after the master has ensured that all safety-critical ECUs are in a trustworthy software state, the master allows the vehicle engine to launch. This way, compromised ECUs are prevented from jeopardizing the safety of the vehicle. To address the heterogeneity of ECUs, we presented two distinct attestation techniques. The first technique targets simple sensor and actuator ECUs, while the second is designed for more complex ECUs, e.g., used for sensor fusion. Since both techniques are applicable to many existing commodity ECUs, our solution entails only minimal deployment costs. We evaluated our scheme in an automotive network that uses CAN and Ethernet. Our results show that the overall timing overhead to verify 100 ECUs is less than 1.4s. Thus, our scheme causes no perceptible delays for passengers.

Acknowledgment

This work has been co-funded by the Federal Ministry of Education and Research of Germany (BMBF) within the UNICARagil project and the Hessen State Ministry for Higher Education, Research and the Arts (HMWK) within CRISP.

References

[1] T. Alves and D. Felton, "TrustZone: Integrated Hardware and Software Security," *ARM Technical White paper*, 2004.
[2] W. A. Arbaugh, D. J. Farber, and J. M. Smith, "A secure and reliable bootstrap architecture," in *IEEE S&P*, 1997.
[3] F. Armknecht, A.-R. Sadeghi, S. Schulz, and C. Wachsmann, "A security framework for the analysis and design of software attestation," in *ACM CCS 2013*.
[4] AUTOSAR, "Specification of Secure Onboard Communication," V.4.3.1.
[5] S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, K. Koscher, A. Czeskis, F. Roesner, T. Kohno *et al.*, "Comprehensive experimental analyses of automotive attack surfaces." in *USENIX Security*, 2011.
[6] K. Eldefrawy, N. Rattanavipanon, and G. Tsudik, "HYDRA: hybrid design for remote attestation (using a formally verified microkernel)," in *ACM WiSec*, 2017.
[7] R. Escherich, I. Ledendecker, C. Schmal, B. Kuhls, C. Grothe, and F. Scharberth, "SHE–Secure Hardware Extension Functional Specification," 2009.
[8] A. Ferraiuolo, A. Baumann, C. Hawblitzel, and B. Parno, "Komodo: Using verification to disentangle secure-enclave hardware from software," in *ACM SOSP*, 2017.
[9] I. Foster, A. Prudhomme, K. Koscher, and S. Savage, "Fast and vulnerable: A story of telematic failures," in *USENIX WOOT*, 2015.
[10] A. Francillon, Q. Nguyen, K. B. Rasmussen, and G. Tsudik, "A minimalist approach to remote attestation," in *DATE*, 2014.
[11] L. Jäger, R. Petri, and A. Fuchs, "Rolling dice: Lightweight remote attestation for cots iot hardware," in *ARES*, 2017.
[12] F. Kohnhäuser and S. Katzenbeisser, "Secure code updates for mesh networked commodity low-end embedded devices," in *ESORICS*, 2016.
[13] X. Kovah, C. Kallenberg, C. Weathers, A. Herzog, M. Albin, and J. Butterworth, "New results for timing-based attestation," in *IEEE S&P*, 2012.
[14] G. Lee, H. Oguma, A. Yoshioka, R. Shigetomi, A. Otsuka, and H. Imai, "Formally verifiable features in embedded vehicular security systems," in *IEEE VNC*, 2009.
[15] Linaro, "Open Portable Trusted Execution Environment," op-tee.org.
[16] C. Miller and C. Valasek, "Remote exploitation of an unaltered passenger vehicle," *Black Hat USA*, 2015.
[17] J. Noorman, P. Agten, W. Daniels, R. Strackx, A. Van Herrewege, C. Huygens, B. Preneel, I. Verbauwhede, and F. Piessens, "Sancus: Low-cost Trustworthy Extensible Networked Devices with a Zero-software Trusted Computing Base," in *USENIX Security*, 2013.
[18] S. Nürnberger and C. Rossow, "–vatiCAN–Vetted, Authenticated CAN Bus," in *CHES*. Springer, 2016.
[19] H. Oguma, A. Yoshioka, M. Nishikawa, R. Shigetomi, A. Otsuka, and H. Imai, "New attestation based security architecture for in-vehicle communication," in *IEEE GLOBECOM*, 2008.
[20] A.-I. Radu and F. D. Garcia, "Leia: A lightweight authentication protocol for can," in *ESORICS*. Springer, 2016.
[21] H. Raj, S. Saroiu, A. Wolman, R. Aigner, J. Cox, P. England, C. Fenner, K. Kinshumann, J. Loeser, D. Mattoon *et al.*, "fTPM: A Software-Only Implementation of a TPM Chip," in *USENIX Security*, 2016.
[22] R. Sailer et al., "Design and Implementation of a TCG-based Integrity Measurement Architecture," in *USENIX Security*, 2004.
[23] S. Schulz, A. Schaller, F. Kohnhäuser, and S. Katzenbeisser, "Boot Attestation: Secure Remote Reporting with Off-The-Shelf IoT Sensors," in *ESORICS*, 2017.
[24] C. Shepherd, R. N. Akram, and K. Markantonakis, "Establishing mutually trusted channels for remote sensing devices with trusted execution environments," in *ARES*, 2017.
[25] J. Van Bulck, J. T. Mühlberg, and F. Piessens, "Vulcan: Efficient component authentication and software isolation for automotive control networks," in *ACM ACSAC*, 2017.
[26] M. Wolf and T. Gendrullis, "Design, implementation, and evaluation of a vehicular hardware security module," in *ICISC*. Springer, 2011.