Decentralized Firmware Attestation for In-Vehicle Networks

Mohammad Khodari, Abhimanyu Rawat, Mikael Asplund, Andrei Gurtov
Dept. of Computer and Information Science, Linköping University, S-581 83
mohkh623@student.liu.se, {abhimanyu.rawat, andrei.gurtov, mikael.asplund} @liu.se

ABSTRACT

Today's vehicles are equipped with a large number of *electronic control units* (ECUs), which control everything from heating to steering and braking. Due to the increasing complexity and interdependency of these units, it has become essential for an ECU to be able to ensure the integrity of the firmware running on other ECU's to guarantee its own correct operation. Existing solutions for firmware attestation uses a centralized approach which means a single point of failure. In this article, we propose and investigate a decentralized firmware attestation scheme for the automotive domain. The basic idea of this scheme is that each ECU can attest the state of those ECU's on which it depends. Two flavors of ECU attestation i.e. parallel and serial solution were designed, implemented and evaluated. The two variants were compared in terms of both detection performance (i.e., the ability to identify unauthorized firmware modifications) and timing performance. Our results show that the proposed scheme is feasible to implement and that the parallel solution showed a significant improvement in timing performance over the serial solution.

KEYWORDS

ECU, attestation, firmware, communication system security, integrity

ACM Reference Format:
Mohammad Khodari, Abhimanyu Rawat, Mikael Asplund, Andrei Gurtov, Dept. of Computer and Information Science, Linköping University, S-581 83 mohkh623@student.liu.se, {abhimanyu.rawat, andrei.gurtov, mikael.asplund} @liu.se . 2019. Decentralized Firmware Attestation for In-Vehicle Networks. In *5th ACM Cyber-Physical System Security Workshop (CPSS '19), July 8, 2019, Auckland, New Zealand*. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3327961.3329529

1 INTRODUCTION

In the vehicle industry we have seen major advancements due to the introduction of electronics and software. Vehicles are controlled by a large number (up to 100 or more) of so called Electronic Control Units (ECUs). ECUs are specialized computers that perform everything from critical tasks such as engine control to less critical functionality such as window control. Many people see vehicles as "computers on wheels" [1] due to how dependent they are on ECUs. A very prominent trend that can be observed today in relation to vehicles is the emergence of self-driving capabilities, which puts even more control of the vehicle in the ECUs. A malfunction of critical components or manipulation of software or hardware in ECUs can cost people their lives. Therefore, it is very important that all ECUs in a vehicle are secure and that the software inside them cannot be tampered with, without it being noticed. Moreover, the ECUs in a vehicle are becoming increasingly more dependent on each other, so that the correct operation of one ECU depends on the correct operation of other ECUs.

An example of what the consequences of having insufficient security can be is the recent hack on the Tesla model X performed by a Chinese research team from Tencent[2]. Even though Tesla introduced firmware code signing to prevent such malicious activity, the team at Tencent managed to hack the vehicle remotely. By exploiting security holes in the car's different ECUs, the researchers managed to open its doors, blink the lights, control in-car display, and surprisingly, engage the breaks whilst the car was in motion. Once such security vulnerabilities have been found, it is very important that OEMs (Original Equipment Manufacturers) update the firmware on their vehicles with a security patch.

ECUs in vehicles used to have the firmware placed in read only memory (ROM), which therefore could not be updated. This is no longer the case since the ECUs in today's vehicles have flash memory. Flash memory allows authorized entities to update or flash a new version of the firmware. This enables fixing known bugs and security holes in the software. By updating the vehicle's firmware to mitigate known vulnerabilities and exploits, the vehicle's firmware can maintain a higher level of security [12, 19].

The ability to update the firmware can also have a negative effect. It will increase the attack surface present for various malicious actors. For example, a malicious actor can try to introduce new malicious firmware on the vehicle or they can try to prevent a new update (which patches some vulnerability) from being installed. Most newer vehicles today have some sort of network connection for receiving firmware updates and other functionalities which can be exploited for this purpose.

In this article we investigate cryptographically secure distributed solutions in order to determine if controllers in a vehicle are in a consistent state. This means that each ECU connected to the vehicle communication bus can attest that all other ECUs in the network are in a consistent/expected state by initiating the proposed solution. Previous solutions to this problem use a centralized approach where a single node is responsible for attesting the state of all other ECUs. This could prove problematic if this particular node is under attack

[1]https://www.theglobeandmail.com/globe-drive/how-cars-have-become-rolling-computers/article29008154/
[2]https://eu.usatoday.com/story/tech/2017/07/28/chinese-group-hacks-tesla-second-year-row/518430001/

or simply fails by some other cause. We propose the use of a fully decentralized solution where each ECU has the ability to attest the firmware of all other ECUs. In practice, not all ECUs will attest all other ECUs since that will result in a quadratic increase of attestations as the ECUs grow. A more reasonable approach is to let those ECUs on which a particular ECU is dependent be attested by it. For example, the breaking system and lights in the vehicle has direct relation, when we hit the break, tail lights should go. In this case the ECU responsible for the breaks and vehicle lights should attest themselves.

To the best of our knowledge, this paper is the first to propose and evaluate a decentralized attestation scheme for automotive systems. We analyze two flavors of the attestation scheme, one where the attestations are performed in series and one where they are parallelized. We implement and evaluate these protocols using real ECUs connected over a Controller Area Network (CAN) bus. Our attestation scheme also takes into account the management of attestation information that needs to be present in all ECUs as well as the firmware update process in which this information must also be updated at each ECU.

To summarize, this paper describes two novel contributions:

- i **New consistency verification mechanisms**: Two newly developed consistency verification mechanisms/solutions specially tailored for the automotive domain are presented, one serial solution and one parallel solution.
- ii **Implementation and testing on real ECUs**: The two proposed consistency verification mechanisms are implemented and tested on real ECUs. The detection and timing performance of the implemented solutions are assessed.

The remainder of this paper is organized as follows. In Section 2 we introduce ECU, CAN and security architecture. Section 3 presents related work on the subject of remote attestation with a focus on the automotive domain. Our decentralized attestation scheme is first presented for a single ECU pair in Section 4, and the two variants for extending to multiple ECUs is described in Section 5. Section 6 describes the evaluation of the attestation scheme, including a comparison of the two multi-ECU flavors. Finally, Section 7 concludes the paper.

2 BACKGROUND

In this section we provide a brief background on the ECU and CAN concepts that are fundamental in the automotive domain, and also briefly describe the Double ratchet key derivation protocol on which the proposed protocols rely.

2.1 Electronic Control Unit

ECU[3] - Engine control Unit takes the sensor data and makes informed decisions using those. ECUs control the units which performs the essential functions such as engine control, break control, air cooling, driver assistance. These days the modern in-vehicle hardware uses multiple ECUs to control the different functions of the vehicles and make it manageable through inter-ECU communication on a CAN bus mostly. Not all the ECUs have the same functional level, they are modified by the nature of the task assigned to the ECU unit.

[3]https://www.bosch-mobility-solutions.com/en/products-and-services/passenger-cars-and-light-commercial-vehicles/steering-systems/electric-power-steering-systems/electronic-control-unit/

2.2 Controller Area Network

CAN is the most commonly used communication protocol in vehicles. This protocol lies on the physical and data link layers on the OSI model. It allows message broadcasting between microcontrollers. CAN was originally designed by Bosch to decrease the complexity of wiring harnesses in vehicles and to enable real-time applications. CAN can be seen as the bus which connects ECUs in vehicles together, enabling cross ECU communication. Furthermore, CAN also allows sharing of sensors between ECUs [4, 11]. Figure 1 represents a CAN bus with different ECUs used in Scania trucks.

CAN is a broadcast protocol, meaning that everything sent over the CAN bus is readily accessible by all CAN connected devices. Therefore, in order for nodes to only react on a specific message, local message filtering is required. Moreover, some error-detection measures exist. A so called Cyclic Redundancy Check (CRC) checksum is included in CAN messages. The checksum is used to ensure the integrity of transmitted messages.

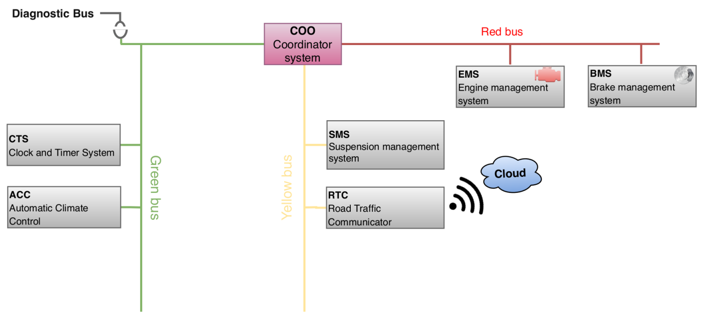

Diagnostic Bus
COO Coordinator system
Red bus
EMS Engine management system
BMS Brake management system
CTS Clock and Timer System
ACC Automatic Climate Control
Green bus
Yellow bus
SMS Suspension management system
RTC Road Traffic Communicator
Cloud

Figure 1: CAN bus architecture in Scania trucks.

2.3 Double Ratchet

Double ratchet[4] is a key derivation protocol that produces two symmetric-key pairs for each communicating party, one for receiving messages and one for sending messages. The produced symmetric-key pairs are used for encryption/decryption or authentication of transmitted messages. For each sent and received message, new sending and receiving symmetric-keys are generated.

What makes this algorithm special is its appealing characteristics:

- **Forward security**: Earlier generated keys cannot be calculated from newer keys. In other words, if an attacker derives a key used for encryption or signing, only that message is compromised, earlier key still seem random to the attacker even though the attacker knows the current key.
- **Backward secrecy**: Newer keys cannot be calculated from earlier keys. In other words, if an attacker derives an previously used key, that key cannot be used to derive later keys.

[4]https://signal.org/docs/specifications/doubleratchet/doubleratchet.pdf

- **Replay-attack resistance**: Because new symmetric key pairs are generated continuously, a replay-attack will not be possible since the key used by the replayed message is not used anymore.

The *Double ratchet* key derivation protocol is used by the proposed solutions presented in this paper for message authentication. This is done by generating message authentication codes (MAC) using symmetric keys provided by *double-ratchet*.

3 RELATED WORK

In the paper [7], several threats against the CAN network are showcased. They provide practical examples on how to manipulate the CAN network for potentially malicious purposes, such as disabling the airbag without triggering the airbag failure indicator on the dashboard. It is thus imperative to be able to know when the integrity of the system has been compromised. This can be achieved by utilizing cryptographicly secure methods such as firmware attestation. Kleberger [9] presented and analyzed several methods for combating malicious activity in a vehicle's communication bus, including authorization protocols for diagnostics.

In the work by Stephan et al. [19], four security classes are defined related to ECU firmware updates. The purpose of these classes is to provide primitive methods of combating firmware manipulation and transmission errors during firmware updates. Methods such as *message authentication code* (MAC) and hash-value authentication are presented.

There is some work related to firmware attestation in the automotive domain, but it is clearly an area in need of more study. The papers by [17] and [22] both present centralized solutions for ECU attestation where one master ECU verifies the other ECUs.

In a paper by Gui et al. [5], whitelisting, blacklisting and a threshold mechanism are used. The complexity of maintaining the states and conditional aspects of the protocol makes it compute-heavy for an IoT device specially where the power could be a constraint. The update mechanism is missing which is highly valuable when a vulnerability is uncovered and OEM patches it through updates.

Tsudik et al. [8] proposed a decentralized approach where not only the detection but also the healing/recovery technique of the infected devices is discussed. The approach works best for the devices loosely connected in a network i.e. non real-time systems where the critical infrastructure and time sensitivity is not involved.

Moroson et al. [14] designed a machine learning model which takes different traffic characteristics to identify the malicious traffic between the devices. This could help in eradicating sources which are putting malicious traffic over the wire and consume network resources.

Open Charge Point Protocol (OCPP) [1] for charging of Electrical Vehicles brings its own potential security challenges. Man-in-the-middle attacks can abuse resource reservation, energy theft or power overloading of charging infrastructure. It could be also a potential entry point for attacks on vehicle's ECUs.

Koscher et al. [10] showcased how to hack a car. Using the OBD-II interface and a segment code, the authors hijacked the most essential components such as breaking system and speedometer. They also tempered with in car accessories such as air conditioning, music system, and, even locking the door while the driver is inside. Reading the CAN bus messages and manipulating them, clearly poses a serious threat to in-vehicle systems.

Miller et al. [13] demonstrated how a car 100 miles away can be hacked by tempering with the ECU's ROM remotely, thus controlling the car's Windows, speedometer, doors.

Regarding the update and ECU memory verification, work by Nilsson et al. [16] discusses how to securely transfer the updated firmware to an ECU and securing the flashing procedure where the attacker can manipulate the update received. The best way to detect it is after the update has been flashed onto the ECU to verify the content of the memory based on a verification code transferred to the vehicle during the update transmission, using a self-verification technique.

Several interesting articles were also found outside the automotive domain [15, 18, 20, 21]. Essentially all the found solutions functioned in the same way with slight differences in the way they tackle the problem. These solutions either have one node dedicated for firmware attestation also known as a verification node [18,20,21] or each ECU attests itself using self-attestation capabilities such as a secure boot [15, 20]. However, both of these solutions are indeed problematic. In the case where the system uses one node for attestation, a single point of failure issue is present. If the verification node is compromised or deactivated, the entire attestation process will stop functioning as intended. In the case of self-attestation, no entity takes the responsibility to verify the integrity of the entire vehicle state. That is, to ensure that each node in the vehicle consists of the right firmware for that particular configuration, indicating that all ECUs will function correctly together. Furthermore, no external communication is used for the attestation process. This means that if the self attestation process is somehow disabled, it will go unnoticed by other nodes.

Thus, a new and more robust solution is needed to be developed which is specially tailored for the automotive domain. The two solutions presented in this article resolve the issues presented above. Each ECU in the communication bus can attest the state of the whole vehicle (or parts of it) independently, thus mitigating the single point of failure. Furthermore, self-attestation without external communication is not used.

4 PROPOSED FIRMWARE ATTESTATION SCHEME(FAS)

In this section, we present the main components of our decentralized firmware attestation scheme. The purpose of the firmware attestation scheme is to measure and verify the state of all interconnected ECUs in a vehicle. By attesting the state of each ECU in the vehicle, it can be determined if the state of the entire vehicle is in a correct/consistent state.

We first provide an overview of how the attestation protocol operates, and in the following subsections we discuss storing attestation information and how to manage initialization and updates of the ECU firmware.

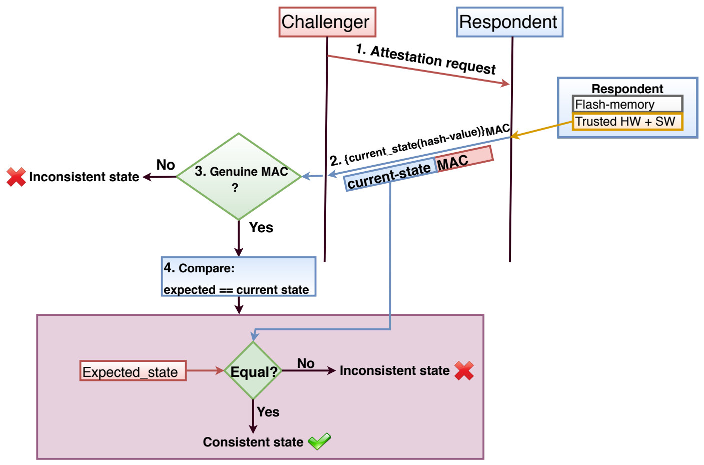


Figure 2: ECU state verification

4.1 Protocol Overview

The firmware attestation scheme is a challenge and response type of protocol. Two main entities are involved in the attestation process, a challenger (the attester) and a respondent (the ECU being attested).

A fundamental assumption in this scheme is the inclusion of a trusted hardware and software extension in each ECU. The reason why a trusted extension is included is to achieve trust between ECUs in a decentralized network. The trusted extension provides secure storage of cryptographic keys and message authentication code (MAC) generation. Furthermore, the trusted extension allows for secure hashing of the ECU's flash-memory in order to provide the current state of the ECU. The attestation process can take place in vehicle in motion or at boot time. Some ECUs such as rain wipers can be turned on by the controlling sensor ECU when it detects rain while the vehicle is still running. Please note that letting the trusted component verify the firmware of its own ECU does not provide sufficient protection since there is no external communication. If the self-attestation process is disabled, it will go unnoticed by other nodes.

In order for one ECU to attest the state of another ECU in the network, four steps are taken as described below. Figure 2 shows a schematic view of these steps.

i The challenger sends an attestation request to the respondent in order to attest the respondent's state.

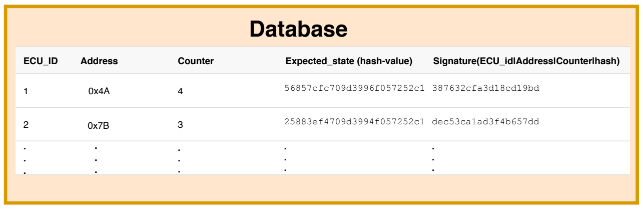
ii The respondent's trusted hardware and software extension will then transmit to the challenger the current state hash-value of the corresponding ECU's flash-memory, accompanied by its corresponding message authentication code (MAC). The MAC provides message integrity and data origin assurance. Moreover, the current state hash-value is the output produced by hashing the respondent's flash-memory. The current state calculation is completely handled and executed by the respondent's trusted extension.

iii The challenger verifies the received MAC. If the MAC is **not** genuine, then the respondent is said to be in an inconsistent state. If the MAC is genuine, the challenger will proceed to step 4. Furthermore, a genuine MAC indicates to the challenger that it can trust the current-state hash-value that was received. This is because the challenger has now verified that the message originated from the trusted hardware and software extension of the respondent.

iv The challenger compares the current state hash-value received to a pre-calculated hash-value, representing the expected state of the respondents ECU. If the current state hash-value is equal to the expected state hash-value the respondent is said to be in a **consistent state** and if they are not equal the respondent is said to be in a **inconsistent state**. If the certain set of ECUs can't trust each other in this case the user or driver of the vehicle can be well notified with the corresponding ECUs functional levels so that user can take manual control.

4.2 Storing attestation information

Recall from section 4.1 that in order to determine if the ECU in question is in a consistent or inconsistent state the current state hash-value and expected state hash-value need to be compared. The challenger obtains the current state from a message transmitted by the respondent's trusted extension. The way ECUs obtain the expected state of other ECUs is by storing them in a local database, stored inside each ECU. Database could reside in a secure hardware space or can be secured via any other mechanism as per the OEMs, hence securing the database is out of the scope of this paper. This is to provide every ECU the ability to gain access to the expected state and therefore the ability to attest other ECUs.


Database

ECU_ID	Address	Counter	Expected_state (hash-value)	Signature(ECU_id|Address|Counter|hash)
1	0x4A	4	56857cfc709d3996f057252c1	387632cfa3d18cd19bd
2	0x7B	3	25883ef4709d3994f057252c1	dec53ca1ad3f4b657dd
⋮	⋮	⋮	⋮	⋮

Figure 3: Information stored in the database

The database in each ECU stores five different fields, *ECU_ID*, *Address*, *Counter*, *Expected_state* and *Signature*, as illustrated in Figure 3 and further elaborated below.

- **ECU_ID**: The *ECU_ID* is the primary key of the database and it is a unique ID used to uniquely identify to which ECU the corresponding row containing the expected state corresponds to. The *ECU_IDs* are only unique within the same vehicle. Each ECU has a global ID meant to be used by the OEM to identify the device globally, meaning that two ECU of the same type, in two different vehicles, have the same *ECU_ID*.
- **Address**: The network address of the ECU belonging to corresponding row. The network address enables transmission of diagnostic attestation requests to the ECU of the corresponding network address.

- **Counter**: The counter is used for replay protection when the update, containing the expected value signed by the OEM, is broadcast. The counter is increased for each received firmware update.
- **Expected_state**: The *Expected_state* is the expected hash-value of the flash memory. It indirectly depicts what state the ECU flash-memory, containing the application, should be in.
- **Signature**: The *Signature* is a cryptographic signature of the concatenation of all the previous fields. This is to ensure each ECU that the expected_state and its meta-data are genuine, not altered and provided by a trusted source, the OEM.

The total required storage space will be dominated by the size of the hash and signature values ($2 \cdot 32$ bytes) multiplied with the number of ECUs. This number will not exceed a few kilobytes at most, making it a lightweight solution.

4.3 ECU Initialization and firmware updates

So far we have described how the protocol operates during normal operation. However, it is just as important to consider how to initialize the states and how to update the firmware of the ECUs while maintaining the proper functionality of the attestation scheme.

4.3.1 Initialization. When ECUs are replaced the database is initialized by sending a *database retrieving* request to the *Update agent* ECU. The *Update agent* is the gateway between a vehicle's internal communication bus and the OEM's back-end services over cellular data. Once the *Update agent* has received the *database retrieving* request it will then respond with the contents of its own database. The request and response messages are signed and authenticated. This operation requires a secure key exchange mechanism such as Double ratchet.

If the *Update agent* itself is replaced, the database content is retrieved from the cloud, since the *Update agent* is the only ECU that has an Internet connection. If in case that the Update agent is not reachable when a new node joins the in-vehicle ECUs, then the database from the existing ECUs can be queried as they maintain the initial database state. The newly joined ECU can validate the update as it has been signed with the OEM key. Its design can be OEM implementation specific as any simple timeout-based solution can be used.

4.3.2 Firmware updates. When updating an ECU's firmware, its expected state will also change. In order for the attestation process to still function properly after an ECU firmware update, the expected state (hash-value) of the corresponding ECU must be updated in each ECU's database. This means that when the update is received from the cloud or from a physical diagnostic session in a repair-shop, the new expected state of the intended ECU, signed by the OEM, must be broadcast in order for all ECUs to update their databases.

Figure 4 shows a visual representation of the update process of the expected state stored in each database. The details of each step are described below.

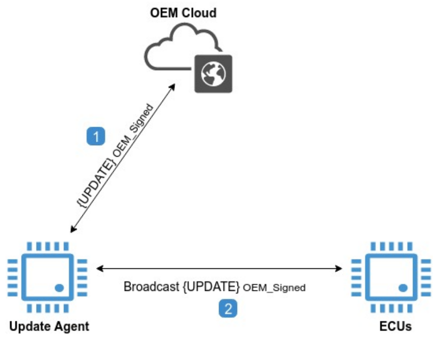


Figure 4: ECU firmware update

i The update is sent from the cloud to the *Update agent* containing the actual update, new expected_state and its corresponding meta-data signed by the OEM, see figure 4. The *Update agent* is the gateway between the internal CAN network and the cellular networks. $Update, \{ECU\_ID, Address, Counter, Expected\_state, signature\}_{sign\_OEM}$

ii The new expected value and its corresponding meta-data are broadcast from the *Update agent* to all connected ECUs publicly. $\{ECU\_ID, Address, Counter, Expected\_state, signature\}_{sign\_OEM}$

iii Upon receiving the broadcast message containing the new expected state of the ECU being updated, the ECU receiving that message stores the message arguments including the signature in their database. The received message is only stored in the database if the *Counter* argument in the update message is larger than the corresponding value in the database. This is to prevent replay attacks of the broadcast message. Furthermore, the message is signed by the OEM such that all ECUs can be assured that its genuine.

$$Counter_{message} > Counter_{DB}$$

5 MULTI-ECU ATTESTATION

The attestation scheme presented in the previous section relates a pair of ECUs, one challenger and one respondent. Now we consider the problem how to attest the firmware for multiple ECUs in the vehicle. We discuss two solutions to this problem, one basic serial version, and one optimized version that performs requests in parallel.

5.1 Serial Attestation

In the serial solution a unique attention request is sent from the attesting ECU (i.e., the ECU that has a need to verify the state of other ECU's) to each and every ECU in the network. The attestation requests are sent serially, one after the other. That is, first an attention request is sent to ECU 1, thereafter ECU 2 and so on. The steps involved in the attestation process are the following:

i The challenger sends an attestation request, indicating that it wants to attest the receiving node, authenticated with a symmetric-key. This is done for every entry in the database.

$$\{attestation\_request\}MAC_{Challenger,Respondent}$$

ii The respondent responds with the current state hash-value of the flash memory provided by the trusted extension of the ECU, authenticated with a symmetric key.

$$\{Current\_state(hash\text{-}value)\}MAC_{Challenger,Respondent}$$

iii The challenger compares the received current state to the expected state stored in the database. If the current and expected states are equal then the ECU in question is in expected state.

$$Expected\_state == Current\_state$$

In several of the steps above, we assume the existence of a shared symmetric key. There are several ways in which such a key can be established (see e.g., the double ratchet method [5] which provides key-establishment and protection from replay attacks), and it is out of scope for this paper to evaluate these mechanisms.

The attester ECU performs the same attestation process for each entry in its database in order to attest that all ECUs in the network are in a consistent state and therefore verifying that the state of the whole vehicle is consistent.

5.2 Parallel Attestation

In the parallel solution a single attestation request is broadcast to all connected ECUs. That is, only one attestation message is sent/broadcast to all connected ECUs. The steps involved in the parallel attestation process are the following:

i The challenger broadcasts an attestation request, indicating that it wants to attest **all** the connected ECUs. The attestation request is sent openly without any cryptographic authentication.

$$attestation\_request$$

ii The respondents, when the broadcast attestation message is received, respond to the challenger with the current state hash-value of their flash memory provided by the trusted extension of the ECU, authenticated with a symmetric key.

$$\{Current\_state(hash-value)\}MAC_{Challenger,Respondent}$$

iii The challenger buffers all the received responses and starts to process the responses one by one. For each response the received current state hash-value is compared to the expected state stored in its database. If the current and expected states are equal then the ECU in question is in expected state.

$$Expected\_state == Current\_state$$

Serial and parallel approaches can also be taken as synchronous and asynchronous solutions. In serial approach, the ECU waits until the current attestation ether completes or fails and then subsequent request is made which is a synchronous nature. On the other hand, in parallel approach the ECU can simultaneously make many requests and collect their responses and act accordingly, therefore behaving like an asynchronous activity. While as some OEMs might want to opt for a hybrid approach for different types of CAN buses and nature of the ECUs based on their function level.

[5]https://signal.org/docs/specifications/doubleratchet/

6 EVALUATION

The purpose of this section is to explain how the attestation scheme (including the two multi-ECU variants) was tested, how the test results were assessed and what the outcome of the tests were.

6.1 Metrics

In order to assess the effectiveness of the proposed solutions, two tests were performed: one test to assess detection performance and the second to assess timing performance. In both tests the number of ECUs was gradually increased until four. The inclusion of four ECUs means that there are one challenger and three respondents. Unfortunately, larger scale tests were not possible due to limited number of ECUs available and high overhead of manual configuration.

6.1.1 Detection performance. The detection performance test is a test that verifies that the *consistency verification mechanism* actually detects tampering of the flash-memory in ECUs/nodes and also detects ECUs with stale firmware.

To assess the detection performance we used the *F1* metric [2]. The F1 metric takes into account true positives, false positives and false negatives in order to produce a value that describes the accuracy of the system under test. The F1 score produces a value scaled between 0 and 1, where 1 corresponds to a perfect result (i.e., no false positives and no false negatives).

6.1.2 Timing performance. The timing performance test is a test that analyzes two different aspects related to the response-time of the attestation process, scalability and predictability. The response-time is the time it takes for the entire attestation process to be performed in its entirety, starting from the moment that the attester engages the attestation process.

- **Scalability**: The purpose of the scalability is to assess if the attestation process is completed within a reasonable time when the number of ECUs are increased. This is done by analyzing the response time of the consistency verification mechanism provided by the timing performance test.

- **Predictability**: The purpose of the predictability test is to evaluate the variation of the response-time of the proposed solution. This is done by calculating the *standard deviation* of the response-times provided by the *timing performance test*. By analyzing the degree of deviation from the mean response-time, it can be determined how close the data set's different data point are to the mean value. The smaller the standard deviation is, the more predictable is the proposed solution's response-time.

6.2 Experimental Setup

This section presents the setting in which attestation solution was tested. Furthermore, the cryptographic functions and key management methods used for the proof-of-concept application are presented.

6.2.1 Proof-of-concept application. To prove that the two serial and parallel solutions actually function as expected, a proof of concept application was developed. This proof of concept application was implemented on real ECUs. The two proposed solutions were implemented on four ECUs with 700 MHz ARM processors. The ECUs in which the solution was implemented on run a Linux based operating system, meaning that regular Linux based programming libraries could be used. The programming language in which the firmware containing the solution was *C++*. tECU The firmware was installed on ECUs using a USB interface which was connected to the ECU in question and the laptop used for programming and testing. The attestation protocol was implemented over the CAN bus at the UDS protocol level. The Unified Diagnostic Services (UDS) is a protocol used for diagnostics of ECUs, it also allows the diagnostic tester (client) to control diagnostic functions in ECUs (servers). This protocol is independent of the physical layer so the application could also operate over an automotive Ethernet network.

The solution was implemented on the UDS protocol level. The reason why it was implemented on such a high level layer, was mainly for two reasons. The first reason was that it is easier to explain the implementation of the solution on a more high level point of view. The second reason is that when OEMs move to using Ethernet instead of CAN, the UDS implementation would still function. The reason why this is the case, is due to the utilization of the UDS protocol for diagnostics over both communication mediums. Furthermore, UDS protocol messages only consist of limited number of elements that are easy to understand. This makes it easier for readers without automotive experience to understand how it all functions and fits together.

6.2.2 Test environment. The test environment was built on a so called plugboard which connects all the ECUs together with a CAN bus. Each ECU's CAN high and low wires were connected to this plugboard. Furthermore, a power-supply was also connected to distribute the electrical power needed.

In order to monitor the CAN messages sent over the CAN bus between ECUs, a USB interface provided by *Vector*[6] was used. This USB interface converts logical CAN messages transmitted over the CAN bus into a digital format that a computer can understand. Moreover, this USB interface can also transmit new messages created on a computer to the CAN bus.

For logging, debugging and testing, a program called CANalyzer was used, which as the name suggests analyzes transmitted CAN messages. The program can log CAN messages and will also segment the messages captured into the parameters that a CAN message consists of. As an example, it can highlight the payload of each message.

6.2.3 UDS protocol. The solutions described in section 5, were implemented using the UDS protocol by utilizing a diagnostic service called *ReadDataByIdentifier*. The *ReadDataByIdentifier* service was used to send attestation requests and receive attestation responses.

6.2.4 Key management. For the timing and detection performance tests we use a simple pre-distributed key. The tests we performed do not include the key distribution phase, so they are not affected by this limitation. Implementing and evaluating the performance of cryptographic mechanisms for key distribution such as Double ratchet on real ECUs is left to future work.

[6]https://vector.com

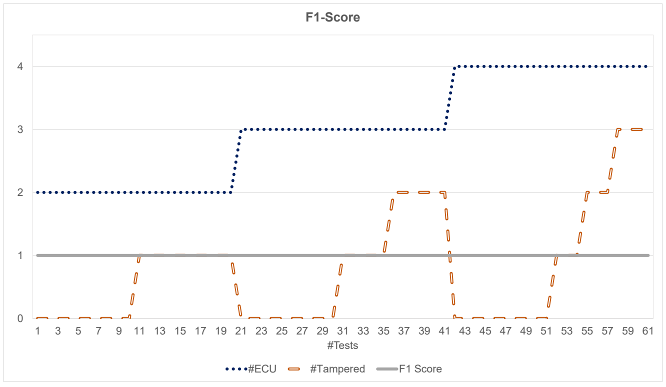


Figure 5: Result of the correctness test.

The reason for this is because *double-ratchet* is a rather complex protocol to implement and test. Furthermore, the implementation that can be found on the *Double-ratchet* homepage utilizes special libraries that are not available on the ECU used for the implementation. Therefore, the only logical outcome was to skip *Double-ratchet* and only have a pre-distributed key used for authentication that does not change.

In each ECU's database, the pre-shared key is also stored. Furthermore, the signatures provided by the OEM that were supposed to be stored inside each ECU's database, were not used.

The reason for this is because for proof-of-concept, testing purposes, the use of signatures inside the database will not change the outcome of the test. The signatures are only used to prove that the content of the database originated from a trusted source, the OEM and are not used in the attestation process. In real life the signature would only be verified at the start, once the database content is loaded into the program and during firmware updates. The use of signature inside the database would not change the response time nor the accuracy of the attestation process.

6.2.5 Cryptographic functions. The cryptographic function used for message authentication was *HMAC-SHA256*.

6.3 Detection Performance

In this section, the results of the detection performance test is presented. The two implemented solutions produced the same detection performance results. The reason for this is due to using the same method for detection in both solution by comparing the current state to the expected state.

Figure 5 shows the results of the F1 score for each iteration of the test, as well as the correlation between the F1 score, number of ECUs and number of tampered ECUs in the test. As it can be seen in figure 5 the F1 score stays at a constant value of 1 during all iterations of the test. The F1 score of 1 indicates perfect accuracy meaning that it can detect tampering of software, parameters and configurations as expected.

In other words, the solution always detects tampering of software, parameters and configurations during testing. Furthermore, the change in number of ECUs included in the test and number of tampered ECUs, do not affect the F1 score. Hence, there is no correlation between the F1 score, number of ECU and number of tampered ECUs.

6.4 Timing Performance

The timing performance test measures the response time between the start and end of the attestation process. Two different properties are analyzed *Scalability* and *Predictability*.

6.4.1 Scalability. Figure 6 presents a visual representation of the scalability test result.

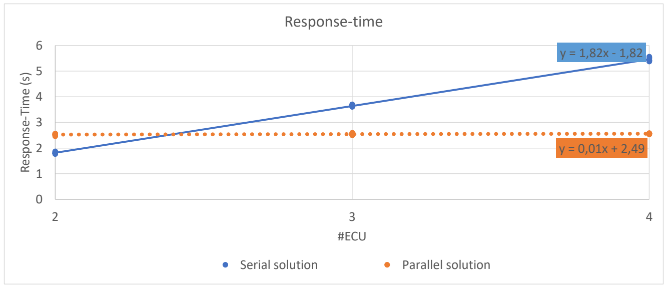


Figure 6: Result of the scalability test.

- **Serial Solution**: It is clear from the figure that the response time of the serial solution increases linearly with the number of ECUs, with a slope of 1.82. Meaning that the response-time increases with approximately 1.8s each time the number of ECUs included in the test is increased. If we consider that today's vehicles can have around 40 ECUs installed, we can estimate how much time it would take to complete the serial attestation process at that scale. As can be seen in figure 6, the following linear equation represents the approximate response-time for each number of ECUs included in the test depicted by variable **x**.

$$y = 1.82(x - 1) \tag{1}$$

By replacing variable **x** with 40 we get the approximate response-time of the attestation process of 40 ECUs. As it turns out, the time it takes to attest 40 ECUs is approximately 71s, more than one minute.

The reason why the serial solution's attestation process takes so much time to complete is because of how diagnostic requests are implemented in the chosen ECU. The UDS protocol is implemented in such a way that several UDS messages cannot be sent at the same time. Every time the attester sends out an attestation request, it needs to wait for the response before sending the next attestation message. The inability to send several diagnostic requests at the same time is a significant bottleneck. If several diagnostic request could be sent directly after each other, which can be done by tweaking the UDS implementation, the increase of the response time for each new ECU would most probably be more similar to the response time of the parallel solution presented in the following section.

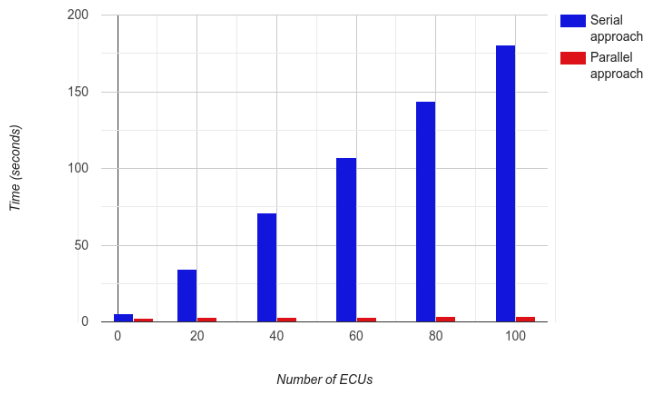


Figure 7: Time comparison of the proposed solutions.

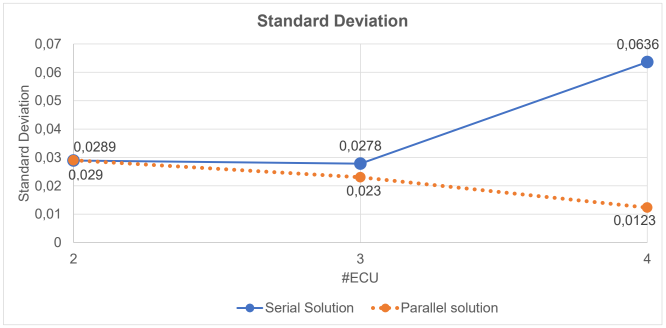


Figure 8: Result of the predictability test.

- **Parallel solution**: The difference in response time when adding one ECU for the parallel solution is approximately 0.01s compared to 1.8s for the serial solution. As can be observed from the diagram, the equation for the trend-line is the following:

$$y = 0.01x + 2.49 \tag{2}$$

If as the serial solution consider that today's vehicles consist of 40 ECUs, we can approximate how much time it would take to attest every ECU in a modern vehicle. By replacing variable **x** in the equation with 40, we get the response-time 2.9s which is a reduction of 96% compared to the serial solution.

The graph in Figure 7 depicts the timing performance of the serial and parallel solutions. We can clearly see that as the number of ECUs increases, the timing performance gap in serial and parallel approach increases. Parallel approach takes less time and thus should be used in certain scenarios where parallel requests are well suited.

6.4.2 Predictability. Figure 8 shows a visual representation of the predictability/standard deviation test result.

From Figure 8 it can be observed that the standard deviation in relation to the serial solution seems to be increasing when the number ECUs is increasing. On the other hand, the standard deviation of the parallel solution is actually decreasing when the number of ECUs included in the test is increasing. Even if the absolute numbers are small, the standard deviation of the serial solution is more than 5 times as high than the parallel solution in case of 4 ECUs.

6.5 Discussion and Future work

By defining the decentralized attestation protocol for in-vehicle network and conducting experiments with real ECU nodes, we see that our proposed architecture successfully works and verifies the protocol functioning. Our experiment deals with four ECUs, which can be considered a small number, but in general a vehicle may contain up to 100 ECUs which is once again not a number where scalability of the solution can be a problem.

We are aware of the fact that some internal implementations are OEM specific, one of which is the update mechanism and ECUs memory management. OEMs use different techniques to update the devices depending upon a number of factors such as, type of memory used on device, implementation related trade-offs, automatic over-the-air(OTA) or manual physical updates. In this paper, we have described an update framework of which the internals can be chosen accordingly, most of the companies try and keep the internals to themselves for security related purposes.

The security mechanism between the ECUs may be further improved with the Host Identity Protocol (HIP) [6]. The future network connectivity of the ECUs with the mainframe provider can be tightened with HIP, which will not only open up faster and more accountable ways to configure the ECUs but also protect against attacks on the update process.

For in-vehicle communication, CAN is widely used in implementations including our paper. CANs use the broadcasting mechanism which is not safe and multiple attacks can be devised easily chocking-up the in-vehicle communication [3]. In future, there can be more robust measures to isolate the current broadcast system to a novel end-to-end or a private network for defined groups of ECUs.

In the attestation protocol described in Section 4.1, upon receiving the attestation request the respondent is sending the hash of the flash memory present on the ECU accompanied by it's MAC. Sending MAC only can very well be used to figure out that whether or not the ECU is being tempered with and then not to communicate with the tempered ECU and report this issue to the user. Sending the original hash can be used to figure out to which part of the ECU is being tempered with by sending multiple hashes of different memory regions. We leave its implementation to the specific OEM providers as they can design this specific mechanism considering their security constraints accordingly.

The attestation protocol works in a Challenger and Responder mode, however during the time of the challenger's request, challenger can attach its hash and MAC to identify itself beforehand so that one cycle of message transmission can be saved and both sides can attest each other in a single RTT. It will also increase the size of the request packet for which the current CAN architecture needs to be modified.

In the proposed solution an attacker can try to exploit the program by manipulating the code working inside the RAM present in the ECU.

So far we have not described the events when an ECU inconsistency is detected. This depends on how critical ECU-controlled system is. For auxiliary subsystems, such as air-conditioning or infotainment, a warning light on the driver's dashboard may suffice. For critical subsystems such as steering or breaks, the vehicle should initiate an automatic graceful showdown, such as slowing down and stopping safely. If the vehicle is in a parked state, it should not naturally even move unless a diagnostic mode is enabled by the car service company.

7 CONCLUSION

Attestation of ECUs is important to prevent accidents to malicious or accidental modification of firmware in vehicles. All existing solutions had only one node designated for the attestation process, also known as the *verification node*. If the *verification node* fails, the entire consistency verification or attestation process would stop functioning. This fundamental flaw called for a new attestation protocol to be developed. Hence, we designed an attestation process that is more robust, where all connected ECUs in the vehicle can by themselves attest the state of the whole vehicle.

We proposed and evaluated two different solutions: a parallel solution and a serial solution. By empirical evaluation of the two proposed solutions, it could be concluded that the parallel solution is more robust and significantly faster than the serial solution. The response time of the serial solution increased by 1.8 sec with each new ECU added to the test, while the parallel solution increased with 0.01 sec. An estimation of the attestation response-time of a network consisting of 40 ECUs based on the trend-line was conducted. It was shown that for the serial solution the response time is approximately 71 sec while for the serial solution the response time would be around 3 sec. That is, a 68 sec difference between the serial and parallel solutions.

The reason for this significant difference in performance is due to implementation restrictions. Moreover, from the F1 score it could be established that both the parallel and serial solutions always functioned as expected by providing the correct ECU state assessment.

ACKNOWLEDGEMENTS

A. Gurtov and A. Rawat were supported by the Center for Industrial Informatics (CENIIT) project 17.01. M. Asplund was supported by RICS: the research centre on Resilient Information and Control Systems (www.rics.se) financed by Swedish Civil Contingencies Agency (MSB).

REFERENCES

[1] C. Alcaraz, J. Lopez, and S. Wolthusen. OCPP protocol: Security threats and challenges. *IEEE Transactions on Smart Grid*, 8(5):2452–2459, Sep. 2017.
[2] S. M. Beitzel. *On understanding and classifying web queries*. Illinois Institute of Technology Chicago, IL, 2006.
[3] P. Carsten, T. R. Andel, M. Yampolskiy, and J. T. McDonald. In-vehicle networks: Attacks, vulnerabilities, and proposed solutions. In *Proceedings of the 10th Annual Cyber and Information Security Research Conference*, CISR '15, pages 1:1–1:8, New York, NY, USA, 2015. ACM.

[4] R. I. Davis, A. Burns, R. J. Bril, and J. J. Lukkien. Controller area network (CAN) schedulability analysis: Refuted, revisited and revised. *Real-Time Systems*, 35(3):239–272, 2007.

[5] Y. Gui, A. S. Siddiqui, and F. Saqib. Hardware based root of trust for electronic control units. In *SoutheastCon 2018*, pages 1–7, April 2018.

[6] A. Gurtov. *Host identity protocol (HIP): towards the secure mobile internet*. John Wiley & Sons, 2008.

[7] T. Hoppe, S. Kiltz, and J. Dittmann. Security threats to automotive can networks–practical examples and selected short-term countermeasures. *Reliability Engineering & System Safety*, 96(1):11–25, 2011.

[8] A. Ibrahim, A.-R. Sadeghi, and G. Tsudik. Healed: Healing & attestation for low-end embedded devices. In *23rd International Conference Financial Cryptography and Data Security (FC 2019), 2019*, February 2019.

[9] P. Kleberger. *On Securing the Connected Car*. PhD thesis, 2015.

[10] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, and S. Savage. Experimental security analysis of a modern automobile. In *2010 IEEE Symposium on Security and Privacy*, pages 447–462, May 2010.

[11] W. Lawrenz and W. Lawrenz. *CAN System Engineering: From Theory to Practical Applications*. Springer-Verlag London, 2 edition, 2013.

[12] K. Lemke, C. Paar, and M. Wolf. *Embedded security in cars*. Springer, 2006.

[13] V. C. Miller C. A survey of remote automotive attack surfaces. BlackHat USA, 2014.

[14] A. G. Morosan and F. Pop. Ocpp security - neural network for detecting malicious traffic. In *Proceedings of the International Conference on Research in Adaptive and Convergent Systems*, RACS '17, pages 190–195, New York, NY, USA, 2017. ACM.

[15] D. K. Nilsson, L. Sun, and T. Nakajima. A framework for self-verification of firmware updates over the air in vehicle ECUs. In *GLOBECOM Workshops, 2008 IEEE*, pages 1–5. IEEE.

[16] D. K. Nilsson, L. Sun, and T. Nakajima. A framework for self-verification of firmware updates over the air in vehicle ecus. In *2008 IEEE Globecom Workshops*, pages 1–5, Nov 2008.

[17] H. Oguma, A. Yoshioka, M. Nishikawa, R. Shigetomi, A. Otsuka, and H. Imai. New attestation based security architecture for in-vehicle communication. In *Global Telecommunications Conference, 2008. IEEE GLOBECOM 2008. IEEE*, pages 1–6. IEEE.

[18] E. Shi, A. Perrig, and L. Van Doorn. Bind: A fine-grained attestation service for secure distributed systems. In *Security and Privacy, 2005 IEEE Symposium on*, pages 154–168. IEEE.

[19] W. Stephan, S. Richter, and M. Muller. *Aspects of secure vehicle software flashing*, pages 17–26. Springer, 2006.

[20] H. Uppal. Enabling trusted distributed control with remote attestation. *Undergraduate Thesis*, 2010.

[21] X. Yang, X. He, W. Yu, J. Lin, R. Li, Q. Yang, and H. Song. Towards a low-cost remote memory attestation for the smart grid. *Sensors*, 15(8):20799–20824, 2015.

[22] Q. Zhou, L. Fei, W. Yi-Huai, and W. Chao. New ECU attestation and encryption mechanism for in-vehicle communication. *DEStech Transactions on Engineering and Technology Research*, (ssme-ist), 2016.