Topic: Security of In-vehicle Communication

Julian Pfeifer

***Abstract*— While IoT devices get more and more widely used, embedded boards are already a central part of every car for several years now. Sensors and embedded boards for car control are connected via a multitude of networks. The most widely used of these networks, the Controller Area Network (CAN), was designed in the eighties with no security in mind because these networks were closed off. But nowadays vehicles have a multitude of interfaces to outside networks, so there is a dire need to modernise the in-vehicle networks with appropriate security measures....tbd**

I. Introduction

The Internet of things (IoT) is a continuously growing network of commonplace devices. They can be physical like wireless sensors and smart phones or virtual like services. The IoT transforms how we interact with our environment. Today almost any electronic device can be bought as a smart version like microwaves, washing machines, light bulbs, door locks and many other ”things”. All these belong to the IoT network. The IoT has several application domains. It can be used personally or in enterprises [19]. The personal and social application domain is for connecting people to their environment and to other people. The industries and enterprises domain enables different activities in and between organizations of for example the finance and banking sector. Also IoT can be used in for example breeding or energy management where it is used for service and utility monitoring. The last application domain is transportation. Which encompasses smart cars and infrastructure like traffic lights.

Ever since there were the first electronic systems to assist drivers in control of their cars there were Electronic Control Units (ECUs) included in these vehicles. ECUs consist of micro-controllers and sensors. Nowadays cars have not only normal Driver Assistance Systems like antilock breaking systems. They also have Advanced Driver Assistance Systems (ADAS) which help people with safety-critical functionality like parking or emergency breaking. To realize all these functions a multitude of in-vehicle networks (IVNs) is needed to connect all the ECUs and the sensors. These IVNs were originally closed off and had no connection to other networks. So they were designed with no security in mind whatsoever. Recently, cars have been connected to a multitude of networks like 3G/4G mobile phone networks, Bluetooth and vehicular ad hoc networks (IEEE 802.11p). So the IVNs have become vulnerable to different types of attacks while joining the IoT. The most used type of IVN today is still the Controller Area Network (CAN). It was introduced in 1983 by the Robert Bosh GmbH. Because in these days IVNs were closed of there were no security features included in the CAN protocol like encryption or message authentication [5]. This absence of inherent security features in the protocol led to successful demonstrations of security breaches [17]. It was possible to reprogram ECUs via physical and wireless connections. The car could even be monitored and controlled remotely. Since then there have been many automotive attack demonstrations [7], [8], [14].

This report focuses on recent CAN authentication research. Two initially proposed authentication protocols named LeiA and vatiCAN secure the CAN network against adversaries that do not control code execution on ECUs in the network [23], [25]. Later these protocols were improved by VulCAN which is an efficient approach to implement secure distributed automotive control software on lightweight trusted computing platforms [26]. VulCAN adds another layer of security by relying on trusted hardware and a minimal software Trusted Computing Base (TCB). Additionally it was shown that VulCAN provides sufficient performance to be used in automotive real-time applications.

In the following chapter II I will provide an overview of different IVNs like CAN, Time-Triggered Networks, Low-Cost Automotive Networks and more. Also I will introduce the general types of security measures in IVNs. Afterwards chapters **??** and **??** will explain LeiA and vatiCAN and how each of them work in general. Chapter V will talk in depth about VulCAN. I will explain how it achieves improvements over LeiA and vatiCAN using Sancus 2.0. Finally there will be a discussion in chapter VI connecting security in IVNs to security in the general IoT sector and the world wide web.

II. In-vehicle Communication Networks and Security

The different functions of a vehicle have very different requirements in performance and safety needs. Therefor the quality of service needed from the communication system varies (e.g. response time or bandwidth). Normally there are different functional domains which divide the in-car embeeded systems [21]. There are the safety-critical domains “powertrain” (e.g. engine control) and “chassis” (e.g. steering) that need a deterministic real-time behavior. The functions of the “body” domain that controls for example dashboard, wipers, lights and windows need to exchange many informations of small size between each other. Other domains like “telematics” and “multimedia” have for example increased requirements in bandwidth and confidentiality.

A multitude of different networks resulted out of this diversity of requirements. Therefor the Society of Automotive Engineers (SAE) created in 1994 a classification for automotive communication protocols. This classification is based on data transmision speed and functionality. There were 4 different classes defined that are labeled class A to class D [4]. Class A networks have a speed lower than 10 kb/s. They are used for convenience features such as trunk release or

electric mirror adjustment. Examples for Class A networks are LIN and TTP/A. Class B has a medium speed of 10 to 125 kb/s. Networks of this classification are for general information between ECUs from for example sensors. Main representatives of this class are J1850 and low-speed CAN. High speed networks of class C have a speed between 125 kb/s and 1 Mb/s and are used for real time control like the power train or vehicle dynamics. High-speed CAN falls into this classification. Above the high speed classification C there is class D. Every communication protocol faster than 1 Mb/s fall into this category. They are normally used either for multimedia applications (e.g. MOST) or for hard real time critical functions like X-by-Wire applications (e.g. TTP/C or FlexRay). Networks this fast, like FlexRay, can also be used as gateways between sub-systems.

In modern vehicles it is normal that there are many networks of different types. A BMW 7 series car from 2008 implements for example multiple LIN buses, a MOST and a FlexRay bus and additionally four CAN buses [16]. All of these networks are normally interconnected by gateways.

In-vehicle networks play a crucial role in keeping the embedded systems in a safe state. Depending on the network it can be for example more or less difficult to identify failed nodes. Also some networks can meet hard real-time constraints and some cannot. In general there are two main paradigms in automotive communication, event-triggered and time-triggered communication [21]. If the communication is consisting of asynchronous events, it follows the event-triggered paradigm. In such systems it is crucial to avoid conflicts while sending events from multiple sources in parallel. Event-triggered systems use the bandwidth effectively and are easy to extend with new network nodes. If network communication is synchronous, so every nodes sends at predefined time slot in a defined interval, then the communication follows the time-triggered paradigm. In general accessing a medium this way is called Time Division Multiple Access (TDMA). These systems are perfectly predictable but require to be statically defined up front. If there are new nodes introduced to the network the schedule has to be changed, so it is more complicated to extend these systems. Also the bandwidth is not used very efficiently and the response times are longer then in event-triggered systems. However missing messages can be identified rather easily. Because both paradigms have up and down sides. Normally both types of communications are needed for different features in an embedded system of a vehicle. Hence some networks like FlexRay provide both types of communication alongside each other.

In the following sub-chapters I will describe the CAN network and give an overview about some of the most representative other networks.

A. CAN Network

The CAN network was introduced in 1983 by the Robert Bosh GmbH [21]. It uses a twisted pair of copper wires as a bus between the nodes. Depending on the speed used it is either classified and used as a SAE class B or class C network. Aside the used speed there are two versions of CAN. Version 2.0A and 2.0B which differ in the length of the identifier used when sending data. CAN uses a Non-Return-to-Zero bit representation with a bit stuffing of 5 bits. Which means the bits will be represented by continuous levels of voltage. To be able to stay in sync when sequences of the same bit value are sent over the wire there will be the other bit value “stuffed” in after every 5 same consecutive values. The receivers know this and can properly “destuff” the bit sequence. So all CAN nodes can stay in sync using this bit stuffing.

For CAN to be able to bound the respond times it uses a priority system. The lower the identifier used to send data the higher the priority. To realize these priorities the physical layer needs to implement an “AND” scheme. Only when everyone simultaneously sends a 1 the resulting bus level is 1. If one node on the bus sends a 0, the resulting bus level has to be a 0. So the 0 is the dominant value on the bus and overrides a 1. With this mechanism it is easy to realize the priority system.

The normally used version of CAN is 2.0A because it provides with 2^{11} possibilities enough identifiers so I will focus on its specs in the following. Communication over the CAN bus is organized into frames of a maximum size of 135 bits if all overheads are included. A frame starts with a 1 bit Start Of Frame. Following this start there is an 18 bit header. The header contains the 11 bit identifier, the Remote Transmission Request (RTR) bit and the Data Length Code (DLC). The RTR distinguished between data frames and data request frames. The DLC provides the length of the data following after the header. This data can have at most 8 bytes. After the data follows a 15 bit Cyclic Redundancy Check (CRC) for ensuring data integrity. After the CRC follows the Acknowledgement field (Ack). With the Ack a sender is possible to know that a sender has received the frame. However it is not possible to distinguish who received it. At the end of the frame is the End Of Frame field followed by the intermission bits. The intermission bits are the minimal number of bits after which it is allowed to send a new data frame.

TODO: image of data frame composition

On an idle CAN bus every node can send a data frame at any time. If multiple nodes want to send a frame simultaneously and collide the priority based arbitration decides which node is allowed to send data. This works with the help of the physical layer implementing the “AND” scheme. So while sending identifier and RTR bit the nodes are observing the bus level. If the node observes a bit of its own to be overridden by a dominant bit (0 is dominant over 1) it stops sending data because another node with a smaller identifier plus RTR is sending at the same moment which has priority. The node which stopped sending needs to wait until the bus becomes idle again and then tries to send the data frame again.

For the priority-based arbitration to work the signal of a bit needs to propagate to all other nodes and back before sending the next bit. So the physical length of the bus restricts the speed with which data can be send. On a 40 meter bus a maximum speed of 1 Mbit/s is possible while only 250 Kbit/s can be achieved over a 250 meter long bus [21]. This restriction has lead to optimizations by the car manufacturers using “traffic shaping”. One example would be to use an offset

for important periodic messages so they will not interfere with each other [20]

The CAN protocol has different mechanisms to detect possible errors. One mechanism would be to use the CRC to validate the data integrity. If an error is detected by a node it will send six consecutive dominant bits which will cancel the current data frame. This will alert all nodes on the bus that an error occurred and a new arbitration phase can start. This delays the message sending and possible deadlines may be missed this way. The possible start of a new frame takes 17 to 31 bits after an error was detected. CAN also includes some fault-confinement mechanisms for failures for example on the hardware level of the micro-controller or communication controller. These mechanisms normally involve the counting of failures and successful delivery of frames. However each node is responsible itself to execute these mechanisms. So the relevance of these mechanisms is questionable [21]. If there is an error for example with the oscilloscope a node could be sending unknowingly many dominant bits. This would be one manifestation of the "babbling idiot" [24]. More mechanisms are needed if the protocol is used for safety-critical functions. There are lots of different solutions for different problems but there is no formal verification for these mechanisms used together [21].

The CAN protocol only defines the physical layer and the Data Link layer but there are higher level protocols like AUTOSAR which use the CAN protocol.

B. Other Representative Networks

There are other priority bus systems besides CAN like J1850 and VAN but they are used less and less in favor of CAN [21]. In 2012 Robert Bosch GmbH introduced a new CAN version named CAN FD [11]. It distinguishes itself from standard CAN through higher data rates outside of the arbitration phase and the possibility for larger data fields. Another advantage is the easy migration from CAN to CAN FD because only the underlying communication layer has to be changed. The nodes using CAN can remain unchanged.

Event-triggered networks such as CAN are not suited for all use cases of IVNs. Event-triggered systems have lightweight protocols with dynamic arbitration per message and good use of bandwidth. However they are hard to verify formally in regard to e.g. bounded response times. If deterministic real time behavior is needed for e.g. X-by-Wire Systems, it is more suitable to choose a time-triggered network based on TDMA (Time Division Multiple Access). They have protocols with static arbitration and need a clock synchronization. The two TDMA based networks suitable for serving as a gateway between IVNs or as a basis for safety critical applications in vehicles are FlexRay and TTEthernet [21].

The FlexRay specification was initially created by the FlexRay consortium which was an alliance of car manufacturers, semiconductor and electronic systems manufacturers (established in 2000). FlexRay is a very flexible network aiming to provide advantages from the time-triggered and the event-driven world by integrating both schemes into one protocol. It is possible to scale the ratio between the time-triggered and the event-driven part of the protocol for adapting the network to the use case. Theoretically it is possible to use just one part of the protocol, the event-driven or time-triggered one.

Besides high speed and real time requirements, there are also parts of the in-vehicle networks with very few requirements in comparison. Their primary requirement is to have low costs. There are a multitude of such networks. The Local Interconnect Network (LIN) and TTP/A are two representatives of these networks whereupon TTP/A is not used in production vehicles and very similar to LIN [21], so I will not go into detail about TTP/A. Low-cost automotive networks are normally used to control seats, doors, etc. They are not only cheap because of the simple communication protocol but also because of the low requirements in regard to micro-controller hardware. LIN is a master/slave network with a single master node polling all the slave nodes for information. The slave nodes are not allowed to initiate communication. Also LIN is able to send nodes to sleep and to wake them up which is very important in the automotive context. It works with just a single copper cable with a data rate up to 20Kbit/s.

The third big IVN domain in vehicles is multimedia and infotainment networks. The de-facto standard in this domain is the Media Oriented System Transport (MOST) [21] developed by the MOST Coorporation [2] which is a consortium of car manufacturers and suppliers. MOST uses Polymer Optical Fiber or coaxial transmission as the physical layer. It integrates the possibility to transport standard ethernet frames. So it is possible to transport content and information for example from the web. The current MOST provides speed up to 150 Mbit/s. However the MOST Coorporation is working on a new version with speed up to 5 Gbit/s to be able to contend with automotive ethernet which is an upcoming competitor.

Today, automotive ethernet is named as the future of in-vehicle networks. However it has still not oust its competitors. Though in 2022 it is likely that there are more ethernet ports inside of vehicles than anywhere else [15]. There are two major representatives of automotive ethernet networks which are Ethernet Audio/Video Bridging (AVB) and TTEthernet. Ethernet AVB works with over-provisioning and prioritisation while the FlexRay competitor TTEthernet is a time-triggered switched ethernet protocol utilizing TDMA. TTEthernet provides mixed-criticality temporal requirements. It describes three types of traffic streams with upper bounds on latency and jitter for two of the three types. TTEthernet was designed for the most critical use-cases even including aeronautic applications. Ethernet AVB was designed for low-latency streaming services over ethernet by a consortium of car manufacturers and suppliers called AVnu Alliance [1].

Automotive ethernet also needs some adjustments of the physical layer to work properly in the vehicular environment in regard to cost, power saving modes, robustness to environmental condition, etc. A physical ethernet layer developed by the OPEN Alliance SIG [3] for the vehicular environment is BroadR-Reach. It uses low cost unshielded copper wires.

C. Security Measures

In-vehicle communication needs to be secured end-to-end so that no one else than the intended receiver can read

the messages, no one is able to modify message content unnoticed and unauthorized parties are not able to participate in the communication [18]. There are three elementary security practices for achieving security in IVN communication. This report focuses on the first elementary practice: Controller Authentication.

1) Controller Authentication: Every valid controller sending messages in the IVN needs to be authenticated. So that it is possible to distinguish messages from unauthorized parties and to be able to process these separately or just discard them. For this to work, every valid controller holds a signed certificate which consists of a controller identifier, a public key and the authorizations for this controller. The responsible gateway possesses a list of trusted Original Equipment Manufacturers (OEMs) public keys. The certificates of the controllers are signed by the respective private keys of the OEMs. This way the gateway can decide which controller has which authorizations and only allow trusted OEM controllers to participate.

2) Encrypted Communication: A fundamental part of secure IVN communication is the encryption of the network buses. Because of the specific hardware constraints in the automotive domain normally a mixture of asymmetric and symmetric encryption is used to meet the requirements on security and performance. Asymmetric encryption could be used to secure the acquisition of the symmetric keys. The symmetric encryption would then be used to secure all bus communication.

3) Gateway Firewalls: Gateways should restrict unauthorized messages to be send into (high safety-relevant) networks. This restriction can work per controller if there are message authentication codes or digital signatures included in each message. If it is technically not possible to include these into each message the only possibility is to grant all controllers of the subnet specific authorizations.

D. Automotive Open System Architecture and compliant CAN Authentication

AUTomotive Open System ARchitecture (AUTOSAR) is a standard created by different vehicle manufacturers, suppliers, service providers and companies from the automotive electronics, semiconductor and software industry to overcome scalability and increasing development costs for ECUs. It specifies a software framework to standardize the ECU software. Since version 4.2 the specification includes security for CAN. It stipulates 128 bit keys and Message Authentication Codes (MACs) of a minimal length of 64 bit. They also specify to use counters or timestamps to provide freshness of the messages. The authentication mechanism needs to be *backwards compatible* to standard CAN to not destroy legacy ECUs that do not need authentication because they are not safety critical.

There are already multiple published CAN authentication protocols. But most of them fail to comply to the AUTOSAR specification. CANAuth [13] and LiBrA-CAN [10] are protocols designed for CAN+ and not CAN itself. They need new CAN transceivers and more powerful ECUs. MaCAN [6] is an authentication protocol designed for CAN and does not need upgraded hardware. However, it splits the CAN payload into 4 byte data and 4 byte MAC. This change in the payload field hinders backward compatibility does not conform to a minimal length of 8 byte for the MAC. LCAP [12] uses many of identifiers and only 2 byte MACs.

To my best knowledge there are only three AUTOSAR-compliant protocols published at the moment: LeiA [25], VatiCAN [23] and VulCAN [26]. While LeiA and VatiCAN are pure software protocols which provide security against the impersonation of protocol participants, the VulCAN protocol enhances these two software protocols with additional system-level security guarantees to provide security against directly compromised ECUs.

III. LeiA: Lightweight Authentication Protocol for CAN

The LeiA [25] protocol was created to authenticate messages in IVNs to prevent attacker from imposing as participants. The authors of LeiA assume that an attacker got access to the network but did not compromise the target ECU. So the attacker is possible to read CAN traffic and send out messages. It uses an unidirectional authentication using symmetric keys. Every participant in the network using the LeiA authentication needs to store information per relevant CAN identifier it wants to send or receive information on:

- CAN identifier
- A long term 128-bit symmetric key used in generating the corresponding session key.
- The epoch is a 56-bit counter which is increased every time the network is restarted or the message counter overflows. It is used in the generation of the session key.
- A 16-bit message counter included in the data frame. It is used for MAC generation.
- A 128-bit session key used for generating the MAC. It is regenerated when the epoch changes to limit the number of messages authenticated per session key.

The authors of LeiA assume that the symmetric keys and identifiers are stored in tamper-resistant memory. So it should require physical access to the vehicle to update these keys.

The session key generation is done at the start to initialise the system. So for every CAN identifier id such a session key is generated by the following steps:

1) increment current epoch value for identifier id by one
2) generate the session key by providing the epoch and symmetric key to the MAC generation algorithm.
3) reset the message counter to zero.

Before sending an authenticated message with LeiA, first the message counter has to be increased. Then the data and the MAC are sent over the corresponding CAN identifier. The receivers will increse its own message counter for the identifier and then verify the received MAC for the data. If this authentication is successful the data is processed (see Fig. 1).

If the authentication of a message fails a resynchronisation between sender and receiver is needed (see Fig. 2). This means they do not use the same session key because the epoch is out of sync. In this case the receiver will send a AUTH_FAIL message to signal the desynchronisation to the original sender

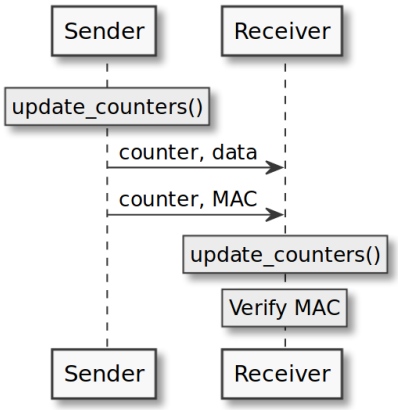

Fig. 1: Sending an authenticated Message using LeiA

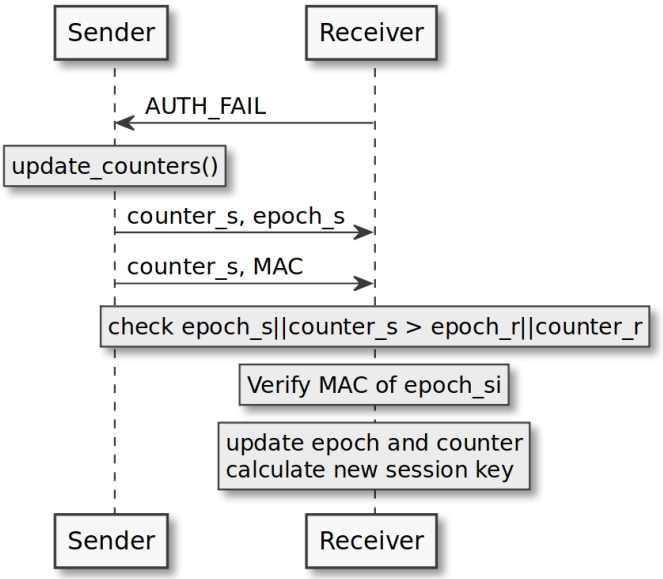

Fig. 2: Message resynchronisation after a failed message authentication between Sender s and Receiver r using LeiA.

of the message causing the authentication problem. The sender will then send out its current epoch value and a corresponding MAC generated from the epoch. The receiver will accept this new epoch and counter value from the sender if:

1) The concatenated value of the received epoch and counter are higher than its own ($epoch_{rec}||counter_{rec} > epoch_{own}||counter_{own}$). This check prevents replay attacks with older messages.
2) The provided MAC for the new epoch is successfully verified with a new generated session key fitting this epoch.

Only if both conditions hold the new received value for epoch and counter are used to update the values of the out of sync receiver. After this the sender will resume normal sending of messages.

In general there is a setup/initiation phase at the start. In this phase the epoch counter will be increased and the message counter will be reset to zero. Additionally the session keys will be generated from the long term symmetric keys and the corresponding epoch value. After the initiation the sending of messages can begin. Before each message the sender will update its message counter. The receiver will update its own counter independently and then verify the provided MAC. If there occurs an authentication error the resynchronization procedure will be triggered. The complete outline of the protocol is given in Figure 3.

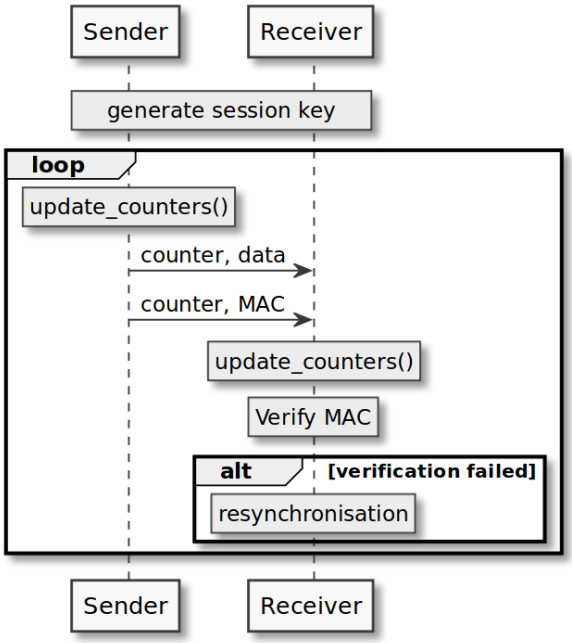

Fig. 3: LeiA protocol outline.

To be able to use an authentication protocol in the in-vehicle network environment it needs to be lightweight. It can not introduce considerable latency or even contain too much code because the ECUs have very limited processing and memory capability. Therefor LeiA uses the MAC algorithm also for generating the session keys so the code can be reused at that point. The session keys are used for compensation of the moderate security provided by the lightweight cryptography. So only a maximum of 2^16 messages are secured by the same session key. After that the epoch has to change and a new session key is used to generate the MACs. In case of a restart of the system the epoch and thus the session key will change too. This restricts the use of a compromised session key.

The LeiA protocol includes the counter in the extended identifier field of the CAN message which is normally used to extend the range of possible identifiers. The first two bits of this field are used to signal which kind of data is included in the payload field: data, MAC of data, epoch or MAC of an epoch. Every ECU using the LeiA protocol needs three communication channels. Therefor it needs three different identifiers. There is a data channel, an authentication channel and an authentication error channel. As seen in Figure 1, every time data is sent there are two messages sent out. One with the data and one with the MAC for the data. The data is sent over the data channel and the MAC is sent over the authentication channel. The AUTH_FAIL messages (see Figure 2)

are sent over the authentication error channel. So every ECU broadcasting messages in the network need to subscribe to all authentication error channels from its subscribers.

If the LeiA protocol is used like that the messages sent roughly double because for every data message a message with a corresponding MAC is sent out. However not all ECUs need the same level of security. So LeiA provides a scaling mechanism by not sending out a MAC for every data message. So for every ECU it is customizable for how many messages MACs are sent out. If there is a low security component there could be only an authentication message sent out for every tenth message.

In [25] the authors show that the LeiA protocol is secure against probabilistic polynomial time adversaries that can freely interact with protocol participants meaning they have only a negligible chance of successfully deceiving one of the participants to have another valid identity. The protocol provides no security against fully compromised ECUs. These possess all the keys they use to broadcast and listen with normally. So a compromised node can listen and broadcast on all these channels and create valid MACs for them. This is an inherent problem of the used symmetric key cryptography. LeiA provides no security against denial of service (DoS) attacks. But it helps by not parsing data from messages with failed authentications.

IV. VatiCAN: Vetted, authenticated CAN bus

The Vetted, authenticated CAN bus protocol (VatiCAN) [23] was designed to provide maximum security while providing full backwards compatibility to standard CAN. VatiCAN provides two core functionalities. The most important functionality is of course the authentication of messages and senders. Secondly, the protocol provides spoof detection for ECUs to detect spoofed messages with their own identifier. Additionally the VatiCAN protocol provides protection against replay attacks. Moreover it defines a changed CAN arbitration to provide spoof prevention.

The authors of VatiCAN define six challenges that a CAN authentication protocol needs to face:

- C1 Since CAN has hard real time constraints, security mechanism for it can only add acceptable communication overhead not causing to significantly increase latency and message collisions.
- C2 Heavy-duty crypto can not be used since ECUs have very limited computational power and memory available.
- C3 CAN messages have only a maximum payload of 8 bytes. Either the messages fit into these 64-bit or the data is broken up into multiple messages.
- C4 Every vehicle must have its own cryptographic keys to not be able to extract working keys from another vehicle.
- C5 Non-critical ECUs should not be modified to be able to keep on using existing hardware. So compatibility is needed with the standard CAN protocol.
- C6 Authenticated CAN messages need to be protected against replay attacks without introducing a global state since CAN is a stateless protocol.

VatiCAN's authors assume that an attacker does not have physical access to the vehicle but compromised an ECU wirelessly. Furthermore they assume that the attacker did not compromise the real target ECU. VatiCAN provides no security against compromised ECUs sending authenticated messages with their own identifiers. So the attacker needs to impersonate another ECU with the help of the compromised one. She is also capable of reading all of the CAN messages and learn about the other ECUs identities.

The message organization in VatiCAN uses separate messages for the authentication. The payload of the original data message is not modified in any way. For the authentication message the next identifier with lower priority is used (id plus one). This assures that both, the data and the authentication message have effectively the same priority with the authentication message not effecting the data message. This design was chosen with challenges C1 and C5 in mind. The receipt of data messages is not influenced by the protocol and unmodified ECUs can still read the data messages without any changes.

For message authentication (concerning C2 and C3) a lightweight key-hashed message authentication code (HMAC) has been chosen. In particular the Keccak algorithm standardized as SHA-3 has been used since it was shown to be the fastest hash function on Atmel embedded microcontrollers [9]. An input to the hash function of 128 bit size is chosen containing the 64 bit data payload plus the identifier for the corresponding data message and a nonce (explained below). So this HMAC is the payload of the authentication message which is sent additionally to the data message as described above.

The symmetric keys used in the hashing function is chosen to be of 128 bit length and injected into the ECUs during assembly (C4). This leads to key updates of multiple ECUs when an ECU is replaced. Gladly ECUs can normally be upgraded through the on-board diagnostics port. So this is no problem and is preferable to a dynamic key-agreement procedure like a Diffie-Hellman key exchange because these are non-trivial to implement on an embedded microcontroller.

To prevent replay attacks (C6) of any kind a nonce is used. These nonce values need to result in different values of HMACs when used in their generation. For every identifier a counter is used as the nonce which is incremented after every message sent. The receiver of an HMAC needs to know the counter value to verify the authenticity. Since the sender and its receivers can get out of sync due to lost messages for example a sychronisation mechanism is needed. This is accomplished by a global Nonce Generator. It periodically sends out new values which all ECUs then use as the next counter value for their identifier after using the current value. So the nonce / counter values are no secret and the attacker can know them without problem. However without the secret symmetric keys for each identifier the nonce is useless because no valid HMACs can be computed without these keys. When two nodes get out of sync in regard to the nonce there is a "deaf" time where no message can be authenticated by the receiver until the next global nonce value is published by the Nonce Generator. See Figure 4 for an overview. The authors of VatiCAN propose an interval of 50 milliseconds for sending

out global nonce values. This would be 1% of the bandwidth of a 500 kbit/s CAN network.

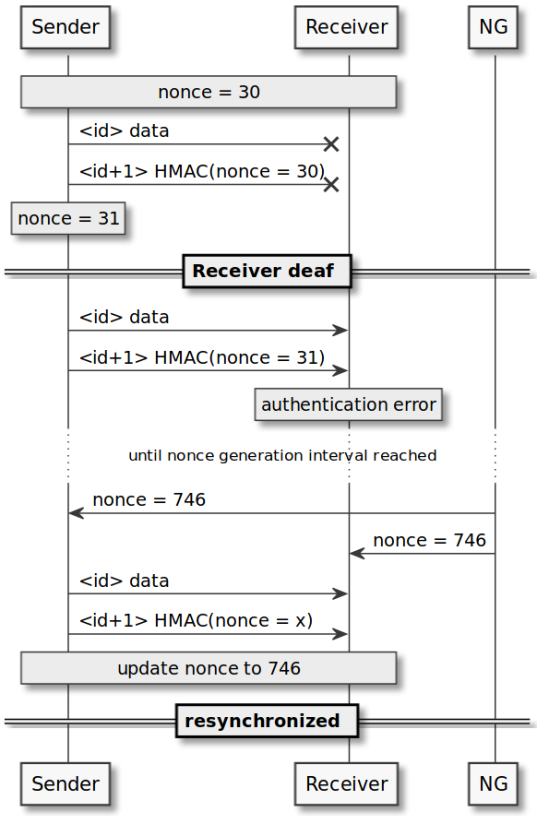

Fig. 4: Nonce Generator sending out new nonce value

Due to the data message arriving before the authentication message, the receiver can start calculating the HMAC of the received payload. So the sender and receiver can calculate the HMAC in parallel and when the HMAC calculated by the sender is received, the receiver can directly compare the HMAC with its own calculation (See Fig. 5)

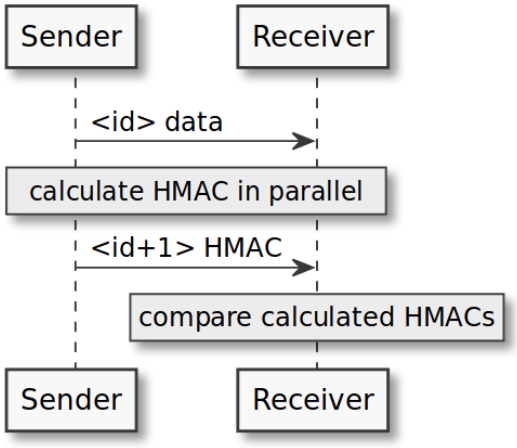

Fig. 5: Sending an authenticated Message using VatiCAN

An ECU is easily able to identify spoofed messages of its own identifier under the prerequisite that every ECU has its own identifier. They can simply look out for received messages with their own identifier. If then the CRC checksum is destroyed with dominant bits while it is sent the whole spoofed message will be dropped by every recipient. Sadly it is not possible to destroy the CRC checksum without modifying the CAN transceiver chip since the CRC is normally handled by the hardware. The authors propose to use such a modified transceiver chip for their added ECU the Nonce Generator because this component has to be added newly anyway for VatiCAN to work. So no attacker will be possible to impersonate the Nonce Generator and control the nonce values of the other ECUs.

The authors of VatiCAN made a Hardware-in-the-Loop-Test with an instrument cluster, accelerator and brake pedals, ATMega microcontrollers and CAN bus controller chips on top of a bench [23]. They implemented VatiCAN as a library for the Arduino development environment for Atmel's AVR microcontrollers. They made a performance evaluation of their protocol and found out that receiving an authenticated message takes 3.3 ms longer than receiving a message without VatiCAN authentication. It was measured on an ATmega 8 bit micro-controller which is the low end of performance for embedded boards in vehicles. The 3.3 ms are not to be underrated. They have an impact of 0.9 m while driving with 100 km/h on a highway.

Van Bulck et al. show in [26] that an advanced replay attack based on the generalized birthday problem (probability theory) is possible against the VatiCAN protocol. This is possible due to the frequent random nonce generation. After recording only 30 minutes of traffic it is possible to start replaying authenticated messages.

V. VulCAN: Vehicular Component Authentication and Software Isolation

VulCAN [26] is in essence no fully defined protocol but defines only the main aspects of one that is largely compatible with every AUTOSAR-compliant protocol. VulCAN extends such a compatible protocol if it is implemented with the VulCAN approach by several system-level security guarantees. The authors call this to "vulcanize" a protocol.

A. Sancus

To achieve this, VulCAN is built on top of Sancus 2.0 [22]. Sancus is a protected module architecture (PMA) that extends the embedded OpenMSP430 processor. It provides a hardware-only TCB extending the memory access logic and processor instruction set. Sancus uses a hardware-level program counter-based access control mechanism which secures the private data section of a software component. So only the responsible code section can access its corresponding private data section. Additionally this code section can only be entered through one specific entry point. With this functionalities it is possible to run multiple mutually distrusting software components in the same address space without being able to spy on each others secrets. It is also possible to build secured driver protected modules (PMs). They can hold exclusive access to Memory-Mapped I/O devices. These driver PMs need to be written in

assembly using only registers to store data because Sancus only provides one contiguous private data section.

Sancus extends the processor with a cryptographic core. This enables hardware-level authenticated encryption, key derivation and key storage functionality. To enable remote / local attestation and secured communication Sancus employs a three level key hierarchy. The root level of this hierarchy consists of a hardware-level node master key. There exists exactly one of these for each embedded computing node that is only known to the owner of this node. Each independent vendor that installs software on this node is assigned a vendor identity. From vendor identity and master key a vendor key can be calculated. These vendor keys build the second level of the hierarchy. Lastly, the third level of the hierarchy consists of module keys for each PM running on the embedded node. The module keys are calculated from the vendor keys plus the corresponding module identity. This module identity contains all content from the code section plus the load addresses of both memory sections of the PM. Upon finishing to load a PM the Sancus-enabled processor will compute the module key and store it in a hardware-level secured memory area. The computed module key can only be used by the respective PM. Independent from this calculation the vendor who possesses also the vendor key and module identity can calculate the module key. So it is easy to check the integrity of the PM with a simple challenge by the vendor using the module key. If the loaded module was modified in any way the module key will not match. This results in the ability to create a secured communication either using the module key or another in-memory key. The module keys calculated by the Sancus hardware also make it possible to distrust the system software running on the embedded nodes. This reduces the TCB to the PMs running on the node which makes formal verifications much easier. Additionally the cryptography core provides hardware primitives for efficient cryptographic calculations which can be used by the PMs. Furthermore Sancus provides secure linking between two PMs on the same embedded node and comes with a modified C compiler automating PM creation and hiding low-level concerns.

B. Vulcanized CAN components

The software-only protocols LeiA [25] and VatiCAN [23] described in the two chapters above are very similar. They use pre-shared symmetric keys between sender and receiver. Additionally both use a monotonically increasing nonce protecting against replay attacks. They generate 64 bit MACs which is then sent in an extra message using the full payload length of this message. This provides compatibility to legacy components because the data messages are untouched. Van Bulck et al. “vulcanized” these two approaches for message authentication to add multiple system-level security guarantees.

VulCAN [26] is an approach which compartmentalizes software into small authenticated software components. These components can run on different or the same ECU. VulCAN supports even mutually distrusting software components to run on the same ECU. Only classical DoS attacks are possible against the vulcanized components. There is no other way to adversely interfere with them. With VulCAN it is not only possible to authenticate the messages sent by the software components its possible to authenticate the integrity of the software components themselves at runtime. This results in the ability to follow an authenticated trusted path an information in the system has taken. From authenticated software component through the insecure network to the next authenticated component and so on. This provides a security that is strongly needed in the vehicular environment where wrong or modified information could have catastrophic consequences.

Attacker Model. The attacker model in VulCAN is mostly similar to the models in LeiA and VatiCAN. But the attacker does not want to impersonate an ECU it wants to instead impersonate a protected component. It is assumed that the attacker gained wireless access to the CAN network through a compromised ECU. The component she wants to impersonate could be on the same ECU or a different one. So the attackers are given the following two capabilities.

Arbitrary Message Manipulation. The attacker has full access to the CAN network. She can read and record all messages sent over the bus, has the right to send own messages with arbitrary payload and identifier and additionally has the possibilities to modify or destroy packets while they pass by.

Arbitrary Code Execution. Formerly presented CAN authentication protocols always assumed the the ECU targeted by the attacker was not compromised itself. VulCAN extends the attacker model so that she is possible to execute and modify code on every ECU. She can even modify privileged system code. The only thing that is protected are the software components secured by Sancus.

Problem Statement. The authors of VulCAN not only describe the protocol requirements for message authentication on an unmodified CAN bus. They also define system-level requirements to establish not only the message authentication but trustworthy component authentication. The protocol requirements are similar to the challenges described in LeiA and VatiCAN:

- **P1: Message authentication.** Message receivers need a strong guarantee that messages are sent by the trusted components of the system.
- **P2: Lightweight Cryptography.** ECUs do not have much computational power so it is necessary to use lightweight cryptography.
- **P3: Replay Attack Resistance.** The protocol needs to be secure against replay attacks despite message loss in the system.
- **P4: Backwards Compatibility.** The authentication system has to use standard CAN transceiver chips and can not impact the functionality of legacy components.

These requirements are already met by LeiA and VatiCAN. But in the face of the wide range of vulnerabilities that ECUs possess [8], [17] the authors of VulCAN recommend strongly to use a more trustworthy solution for authentication in CAN networks that additionally fulfills the following system-level requirements:

- **S1: Real-Time Compliance.** In the vehicular context software hast strict real-time constraints with critical consequences when broken. While DoS attacks are out of scope, the normal operation should not violate these constraints.
- **S2: Component Isolation.** Trusted software components need to be protected against code manipulation so that for example the authentication mechanism can not be modified.
- **S3: Component Attestation.** The integrity of all trusted components and their isolation (S2) needs to be verifiable at the system start to ensure proper operation of the vehicle.
- **S4: Dynamic Key Update.** ECUs need to be replaceable and the system needs to be able to provision new keys for replaced components at runtime. The uninvolved components should not be affected by this.
- **S5: Secure Legacy ECU Integration.** Legacy ECUs should not only be able to work unaffected by the authentication system. The system has to be able to shield legacy ECUs with critical tasks. This allows a slow transition from legacy ECUs to secure and trustworthy components.

C. Authenticated CAN Bus.

Message authentication (P1–P4) follows the same approach as in LeiA and VatiCAN. Messages will be sent out in clear text. Afterwards the authentication for this message will be sent out on another identifier. If available the next highest identifier in respect to the data message will be chosen. Because of the priority inversion in CAN this results in the identifier with next lowest priority. This results in both, data and authentication message, having the same priority in respect to other messages. If this identifier is not available for the authentication messages because legacy ECUs use it then another identifier has to be chosen (P4). The payload of the authentication message consists of a 64 bit MAC. A 128 bit symmetric key (P1, P2) is used to calculate the MAC from the data payload, the data identifier and a monotonically increasing counter to protect against replay attacks (P3). Every identifier is assigned one symmetric key and it is allowed to use the same keys for multiple identifiers to save used memory (P2).

It is very important that the MACs are calculated efficiently to comply with the real time constraints of the vehicular domain (S1). The LeiA protocol settles for using symmetric cryptography. VatiCAN, however, computes the MAC at sender and receiver in parallel. This is possible because the data message is sent first. Even if this roughly halves the time for authentication it still takes a few milliseconds (see Chapter IV) to calculate a MAC with software only on low-end ECUs. VulCAN instead uses the hardware-level authenticated encryption primitives provided by Sancus [22]. For this the Sancus cores are configured to use 128 bits of security and the result is then truncated to 64 bit by discarding the eight least significant bytes of the result. The use of hardware for calculating the MACs in VulCAN reduces the time needed by a whole magnitude (see !TODO??!).

The Nonce value included in the MAC calculation for replay attack resistance (P3) is needed to be initialized in different situations. This is necessary on different occasions because no nonce value should be used with a key more than once to ensure security (P3). It is needed every time the platform resets or the value overflows and when sender and receiver get out of sync due to lost messages. This challenge is commonly addressed by using short-term session keys which are calculated from the long-term pre-shared key and the help of an epoch counter. Every time a new session key is used the nonce can safely start from zero. This simplifies the nonce initialization problem to securely storing a long-term pre-shared key and an epoch counter. The finally proposed key provisioning system (S4) in VulCAN further relaxes this requirement to store the session keys and epoch values securely on *one* ECU (See Chapter V-D).

To use a new session key and reset the nonce to zero solves the problem of nonce initialization when a restart or nonce value overflow occurs. But in the case of message loss this is no reasonable solution. Due to high network load or just ECUs switching to standby it is very common that broadcasted messages are not received by all or any of the receivers. The sender will continue to increase the nonce while the receivers missing the messages can not know the current value of the nonce. So a nonce resynchronization is needed between sender and receiver. Actually, the nonce value is no secret and could be included into the data or authentication message. VulCAN makes no strict requirement how to resynchronize the nonce and leaves it the underlying authentication protocol. It would be possible to include the nonce into the data message when not all of the payload field is needed for the data. The authentication message payload is already filled with the MAC so there is no room for the nonce value there. LeiA includes the nonce value into the extended identifier field of the CAN frame as described in Chapter III. This could produce problems with legacy modules using the extended identifier field. VatiCAN uses a special component called the Nonce Generator to frequently send new global nonce values. The VulCAN authors showed in [26] that this approach is not secure against advanced replay attacks. This is described in detail in Chapter IV.

D. Component Isolation and Authentication

Key storage (S2) is the most critical part of an authentication system. The previous presented authentication protocols [6], [12], [17], [23], [25] always assumed that the targeted ECU is not compromised itself so its key has not leaked to the attacker. VulCAN instead considers an attacker who can modify code on every ECU. So every key that is not explicitly protected will be known to her. To achieve the needed protection against an adversary with so much freedom, VulCAN uses Sancus' PMs to secure the authentication protocol and its private data. This reduces the TCB on the ECUs. It is not even necessary to trust the system management software on the ECUs like the privileged kernel. If a message is received it is possible to trace back the trusted path (trusted software components protected by PMs and authenticated messages) from where the data came. This reaches all the way back to sensor drivers which can also be secured by Sancus' PMs. This reduction of

the TCB makes verifications, experimental or formal, of the system much easier.

The software components are only trusted after proper isolation and provisioning with the symmetric keys. After each restart all ECUs can be controlled completely by an attacker. So the loading of the software components is done by untrusted system software. This leads to the following three challenges for achieving this wished state after the loading phase:

1) Attesting the integrity of the loaded PMs
2) Provision the session keys over the untrusted CAN bus
3) Let the ECUs be replaced by an untrusted third party

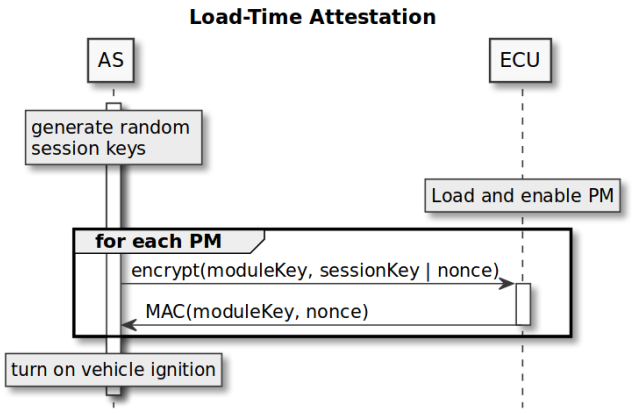

Fig. 6: Load-time attestation (after [26])

For attesting the integrity of all loaded PMs (S3) the VulCAN protocol designed an Attestation Server (AS) that conducts a remote attestation of all loaded PMs. To be able to do that the AS needs the Sancus module keys for every PM. With these it is easy to verify the integrity of each PM. A simple challenge attestation using the module key will suffice. The challenge should contain an nonce that monotonically increases to protect against replay attacks (P3). On top of that, the AS can additionally provision session keys to the PMs while attesting their integrity. This can be done by sending a random session key plus the nonce encrypted by the respective Sancus module key to the PM. Only if the PM is unmodified it can properly decrypt the session key and the nonce. Then it can send back a valid MAC of the nonce to complete the attestation.

This solves the first two challenges but leaves open how to replace an ECU by an untrusted third party (S4). To achieve this we need to use the higher key hierarchies from Sancus (see Chapter V-A). The remote car vendor functions as the trusted infrastructure provider, it knows all root node keys of the ECUs. It is possible to independently calculate all Sancus keys for its ECUs. Additionally we need the AS to possess an asymmetric key. Whenever a new component is installed into the vehicle it sends a unique replacement identifier. The AS receives this identifier and sends it to the remote car vendor. Together with the identifier, the AS needs to send a certificate of its public key signed by the vendor. If the vendor positively verifies the certificate he sends the module key of the newly

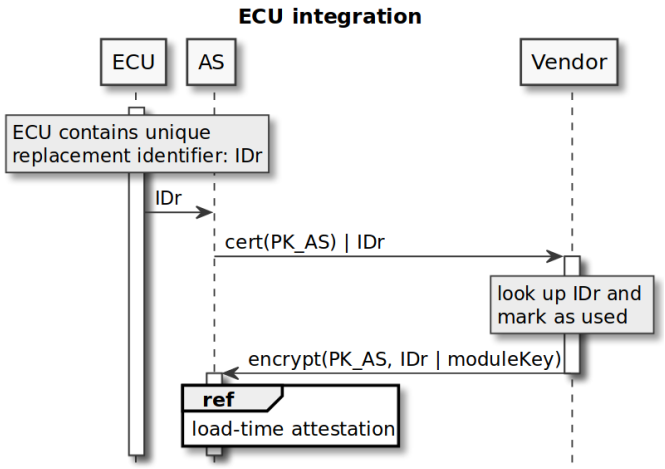

Fig. 7: Integration of a new ECU (after [26])

installed component to the AS encrypted with the public key of AS. This way only AS can decrypt the sent module key of the new component. Subsequently, the normal attestation protocol can be executed for the new component.

VulCAN introduces not only new software but also depends on new hardware as well in contrast to other introduced solutions like LeiA [25] and VatiCAN [23]. However, the pure system-level security guarantees provided by the Sancus hardware are not the only benefits gained. With the Sancus hardware it is possible to securely shield legacy ECUs (S5). If a security gateway is set in front of the legacy ECU to seperate it from the rest of the network, the gateway can perform all the necessary authentication as if it were the legacy ECU. Due to the separate data and authentication messages this is easily achievable. To have Gateways between CAN networks in the vehicle network is not uncommon. But the commonly used gateways have the problem of being insecure to exploitation or reprogramming of their software [8]. If we remember, the attacker model of VulCAN allows it to reprogram every ECU in the network which is not explicitly protected against it. However, due to the software isolation feature of Sancus it is possible to secure the integrity of the gateway and so properly secure the legacy CAN network behind the gateway. The prerequisite for this to work is the implementation of the gateway CAN driver as a Sancus PM. VulCANs authors think this is acceptable because the gateway driver is considerably simplified in contrast to a standard CAN driver.

E. Security

The VulCAN authors analyzed the security of their protocol [26] with the following conclusions.

The VulCAN protocol-level message authentication (P1) and replay attack protection (P3) is safe against adversaries that control the network but not the software on the ECUs. It is infeasible to brute force the session key (out of 2^{127} possibilities). Otherwise she would have to guess a correct combination of identifier, nonce and MAC. Due to the non applicability of the birthday paradox to this problem the attacker would have to guess the correct MAC (there are 2^{63}

different MACs) which is also not reasonably executable in the limited time of the session key validity.

VI. Discussion

Key trade-offs & considerations on the presented technologies for in-vehicle communication security: VulCAN vs VatiCAN vs LeiA

maybe comparison to the internet world with tls etc. to set everything into context

VII. Conclusions

Wrapping up the presented technologies for in vehicle communication, especially VulCAN.

References

[1] Avnu Alliance.

[2] MOST Cooperation.

[3] Open Alliance SIG.

[4] F. Ali, Z. Sheng, and V. Ocheri. A Survey of Automotive Networking Applications and Protocols. Technical report, 2017.

[5] O. Avatefipour and H. Milik. State-of-the-Art Survey on In-Vehicle Network Communication "CAN-Bus" Security and Vulnerabilities. *International Journal of Computer Science and Network*, 6(6):720–727, 2017.

[6] A. Bruni, M. Sojka, F. Nielson, and H. Riis Nielson. Formal Security Analysis of the MaCAN Protocol. pages 241–255. Springer, Cham, 2014.

[7] M. Cheah, S. A. Shaikh, O. Haas, and A. Ruddle. Towards a systematic security evaluation of the automotive Bluetooth interface. *Vehicular Communications*, 9:8–18, jul 2017.

[8] S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, K. Koscher, A. Czeskis, F. Roesner, and T. Kohno. Comprehensive Experimental Analyses of Automotive Attack Surfaces. In *Proceedings of the 20th USENIX Security Symposium*, pages 77–92, 2011.

[9] T. Eisenbarth, Z. Gong, T. Güneysu, S. Heyse, S. Indesteege, S. Kerckhof, F. Koeune, T. Nad, T. Plos, F. Regazzoni, F.-X. Standaert, and L. van Oldeneel tot Oldenzeel. Compact Implementation and Performance Evaluation of Block Ciphers in ATtiny Devices. pages 172–187. Springer, Berlin, Heidelberg, 2012.

[10] B. Groza, S. Murvay, A. van Herrewege, and I. Verbauwhede. LiBrA-CAN: A Lightweight Broadcast Authentication Protocol for Controller Area Networks. pages 185–200. Springer, Berlin, Heidelberg, dec 2012.

[11] F. Hartwich and F. Hartwich. CAN with Flexible Data-Rate. *ICC 2012 – CAN IN AUTOMATION*, 2012.

[12] A. Hazem and H. A. H. Fahmy. LCAP - A Lightweight CAN Authentication Protocol for Securing In-Vehicle Networks. *10th ESCAR Europe*, 2012.

[13] A. V. Herrewege, D. Singelee, and I. Verbauwhede. CANAuth - A Simple, Backward Compatible Broadcast Authentication Protocol for CAN bus. *ECRYPT Workshop on Lightweight Cryptography*, (November 2011):299–235, 2011.

[14] 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, jan 2011.

[15] Ixia. Automotive Ethernet: An Overview. (May):20, 2014.

[16] H. Kellermann, G. Németh, J. Kostelezky, K. L. Barbehön, F. El-Dwaik, and L. Hochmuth. Electrical and Electronic System Architecture. *ATZextra worldwide*, 13(8):30–37, nov 2008.

[17] 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. IEEE, 2010.

[18] K. Lemke, C. Paar, and M. Wolf. *Embedded Security in Cars Securing Current and Future Automotive IT Applications*. 2006.

[19] R. Mahmoud, T. Yousuf, F. Aloul, and I. Zualkernan. Internet of things (IoT) security: Current status, challenges and prospective measures. In *2015 10th International Conference for Internet Technology and Secured Transactions (ICITST)*, pages 336–341. IEEE, dec 2015.

[20] N. Navet and F. Simonot-Lion. Scheduling Messages with Offsets on Controller Area Network: A Major Performance Boost. In *Automotive Embedded Systems Handbook*, chapter 14.1. 2009.

[21] N. Navet and F. Simonot-Lion. In-vehicle communication networks: A historical perspective and review. *Industrial Communication Technology Handbook, Second Edition*, 2013.

[22] J. Noorman, J. O. Van Bulck, J. T. Mühlberg, F. Piessens, P. Maene, B. Preneel, I. Verbauwhede, Iminds-Cosic, K. U. Leuven, J. G. Otzfried, M. Uller, F. Freiling, and F. Erlangen-Nürnberg. Sancus 2.0: A Low-Cost Security Architecture for IoT Devices. *ACM Transactions on Privacy and Security*, 0(0), 2017.

[23] S. Nürnberger and C. Rossow. vatiCAN: Vetted, authenticated CAN bus. In *Cryptographic Hardware and Embedded Systems – CHES 2016. CHES 2016. Lecture Notes in Computer Science*, volume 9813 LNCS, pages 106–124. Springer, Berlin, Heidelberg, 2016.

[24] J. Pimentel, J. Proenza, L. Almeida, G. Rodriguez-Navas, M. Barranco, and J. Ferreira. Dependable Automotive CAN Networks. In *Automotive Embedded Systems Handbook*, chapter 6. 2009.

[25] A. I. Radu and F. D. Garcia. LeiA: A lightweight authentication protocol for CAN. In *Computer Security – ESORICS 2016. ESORICS 2016. Lecture Notes in Computer Science*, volume 9879 LNCS, pages 283–300. Springer, Cham, 2016.

[26] J. Van Bulck, J. T. Mühlberg, and F. Piessens. VulCAN: Efficient Component Authentication and Software Isolation for Automotive Control Networks. *33rd Annual Computer Security Applications Conference*, pages 225–237, 2017.

Appendix