Security in Automotive Networks: Controller/Message Authentication

Manish Mahesh Dalvi
Matriculation Number [redacted]
Email [redacted]

***Abstract*—With the advancement in communication technology and its diversity, the vehicular communication has become smarter and is now connected to the Internet via various protocols. Due to this connectivity, vehicles have been under many attacks which has led to safety concerns among passengers and civilians. While there are many efforts to increase the security of the devices from being subjected to attack, still there is huge scope of improvement since most of the protocols are not available for testing due to Intellectual property rights of the company hence not made public. In this paper we discuss two different solutions that are available which help us establish secure communication between the devices over existing CAN network. First solution is based on software encryption where sharing symmetric keys between group of nodes [1] shows prominent results among rest other software encryption methods on CAN-FD and FlexRay. Second solution being hardware oriented encryption vulCAN [2] - a generic design for efficient vehicle message authentication, software component attestation and isolation using lightweight trusted technology. The solutions adhere to real-time deadlines under amiable conditions.**

I. Introduction

Modern Vehicles come with many sensors and actuators which are in-turn controlled by ECUs. Due to increase in comfort and safety there has been a tremendous increase in the number of ECUs in every vehicle. The ECUs tasks has been diversified as it implements many different functions including hard real-time tasks such as steering and braking or soft real time systems like infotainment system. As the number of ECUs increased, the network became more complex and features like connectivity to the internet were added. ECUs are connected via cables and wireless devices like WiFi, Bluetooth, 3G etc which increase the probability of attacks on the vehicle [3], [4], [5], [6], [7], [8], [9]. This leads to lack of security and compromises the safety of the passenger as well the people around the vehicle. There are numerous examples where the vehicles wiper, braking, engine ignition have been controlled by an attacker and the passengers life was at stake. The industry has been quick in responding to the attacks and has brought in certain level of security but they have not been tested extensively due the inaccessibility of the architectural design.

This paper discusses two solutions which secure the network and also maintain the real time deadlines for safety critical applications.

A. *Software Keying Proposals*

There have been numerous software keying proposals like Voting scheme, CANAuth, MaCAN etc. These deal with different types of keying techniques such as single authentication key, pairwise keys or group keys. Each of these proposals were tested on FlexRay and CAN-FD network over a CANoe simulator with similar conditions to draw a more realistic perspective on the software proposal for bus authentication. CANoe is an industry standard simulation tool which simulated a realistic network based on FlexRay and CAN-FD. The results suggests that group keying along with symmetric keying is by far the most pragmatic solution as there is good trade off between bandwidth, computation load and security offered for the network.

B. *vulCAN*

vulCAN is a security solution that deals with higher level security attacks that have possibility of controlling the entire vehicle ecosystem. vulCAN implements a secure distributed automotive control software with a lightweight cryptography and small trusted computing base. The approach relies on hardware enforced memory protection which isolate critical real time softwares on the ECUs and prevent code-abuse attacks using return oriented programming [10]. vulCAN further shows how to securely interface such ”vulcanised” software components and dynamically attest the components integrity across a non-secure automotive network. There is hardware cryptography involved which reduces the time to encrypt significantly compared to software encryption, that gives way to a secure message authentication on the CAN bus which is necessary to meet real time deadlines.

vulCAN is based on Sancus [11], an open source protected module architecture [12] that extends embedded openMSP430 [13]. vulCAN solution is implemented and evaluated on two AUTOSAR [14] compliant CAN authentication protocols namely vatiCAN [15] and LeiA [16]. vulCAN solution in open source and can be accessed on https://distrinet.cs.kuleuven.be/software/vulcan. The main purpose of vulCAN is to

- Improve software security in the current automotive network using trusted computing base, avoid different types of attacks such as code abuse attacks, replay attacks.
- Provide how to securely keep keys hidden from attacker, key management and security gateway for the legacy ECU devices.
- Give an idea about how vulCAN can be used in future scenarios.

II. Background

A. *Software Keying - Flexray and CAN-FD*

Flexray [17] and CAN-FD [18] have been chosen since CAN is the most common vehicular bus and these two are the successors of CAN. CAN-FD is cheaper than FlexRay but FlexRay is faster and thus FlexRay is used in safety critical Applications.

CAN [19] is the most commonly used broadcast communication network in automotive industry. It was developed in 1983, when there was no possibility of remote attacks as the automotive vehicles were not connected to the internet nor did they have considerable wireless devices. CAN uses fixed data transmission and has one bit which is toggled while receiving an acknowledgement. Attackers with access to ECUs that are connected to this network can easily read and spoof unwanted messages since there is no security provided by CAN.

FlexRay [17] is an automotive communication protocol which was developed by FlexRay Consortium. It was designed as a successor to CAN protocol and as a faster and more reliable network. This protocol supports data rates upto 10Mbps and has payload length of 254 bytes. The data is transmitted in frames which has been shown in Figure 1. FlexRay frames can be time or event triggered and also uses both static and dynamic segment by having pre-defined communication cycle. The communication cycle has an additional symbol window which uses Network Idle time for synchronisation between frames. In the current software keying proposals all the FlexRay frames are configured to static frames and hence sent cyclically with variable recurrences. FlexRay is mainly used in safety critical applications like power train where there is hard real time deadlines to be met.

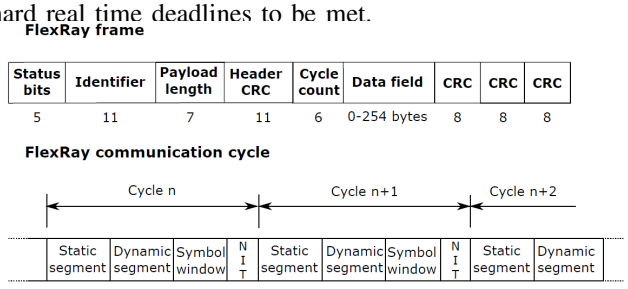

Figure 1. FlexRay

CAN-FD (CAN with Flexible Data Rate) [18] is a communication protocol designed as a successor to CAN due to a demand in higher bandwidth. It was designed by Robert Bosch Gmbh. Figure 2 shows the difference between a CAN and a CAN-FD frame. CAN-FD supports up-to 64 bytes of payload where as CAN supports only 8 bytes. CAN-FD has a Bit Rate Switch(BRS) bit which is used to change between a data rate of 1Mbps to 8 Mbps. CAN-FD has a extended data length bit which is used to make sure that CAN-FD is backward compatible with the original CAN Frame. CAN-FD recently has been given ISO standard with minor changes of CRC field in-order to detect and correct communication errors in the frame during transmission. CAN-FD messages are sent in timely manner between 5ms to 80ms.

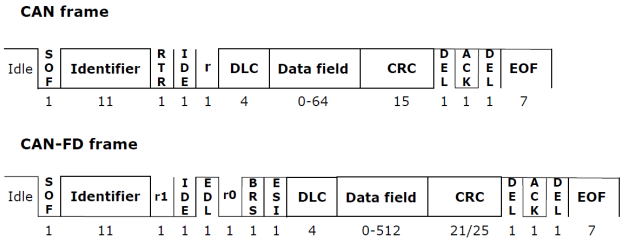

Figure 2. CAN-FD Frame

B. *vulCAN - Embedded Protected Module Architecture background*

Security Primitives. Embedded Protected Module Architecture is the basis of the hardware encryption and also provides hardware level security to software and cryptographic keys. The vulCAN provides security to CAN messages by encrypting them using the hardware enforced encryption. It authenticates and also isolates software critical components in the network by remote attestation. VulCAN builds on the trusted computing features that have been provided by Protected Module Architectures (PMAs), which are a set of new architectures that support secure and isolated execution of critical modules using minimal TCB. Protected modules provide strong confidentiality and integrity guarantees regarding the internal memory and states without trusting any software that is running over it, not even privileged operating system code. Modern PMAs have number of security features namely (i) isolate PM software protection domains, (ii) attest the integrity of an isolated PM, (iii) facilitate key management for secure communication. These features are provided by *Sancus* architecture. VulCAN can be implemented on any platform that provides these security features.

Sancus is a open source embbeded PMA and hardware only TCB that provides memory access logic and instruction set of low-cost, low-power openMSP430 micro-controller. Sancus support multiple mutually distrusting software components that each has contiguous memory sections in a single-share-space. A hardware level program counter ensures that PM's private data section can be accessed by its corresponding code section only, which can be only entered through a single entry point. Sancus provides secure driver PMs with exclusive ownership over Memory Mapped I/O peripheral devices that are through address space.

Key Management in Sancus. Sancus also provides hardware level encryption, key derivation and key management functionality by having a cryptographic core. Sancus offers three level key hierarchy. Firstly, a unique hardware-level *node master key* which is known only to the manufacturer by which the manufacturer secures a part of the protected module (PM) like the cryptographic keys which need to be kept safe. Secondly, *Vendor keys* which are derived from the *Node Master key*. *Vendor keys* are keys for each vendor who is trusted by the manufacturer and is going to install Protected Module on a particular node using the key, which encapsulates the module and does not allow non trusted software components to change any data in the protected memory. Thirdly, *Module keys* are

derived from a vendor key using module identity, which has the contents of a PM's code section and load addresses of both its sections. Sancus enabled processor computes module key and stores it in a hardware-level protected storage area for exclusive access by PM. The symmetric module key can used to establish confidentiality and integrity preserving communication channel between remote software vendor and newly deployed PM. Sancus supports secure linking of two protected modules which are in the same computing platform by means of caller/callee authentication. PMs can benefit from sancus efficient cryptograhic hardware by means of dedicated processor for authenticated encryption/decryption using module or in-memory keys.

III. Proposed Security Solutions

This section gives a brief overview of the two solutions that provide security to automotive networks namely (i) Software Level Encryption by Group Symmetric Keying and (ii) vul-CAN which is a hardware oriented solution and more secure compared to the software proposals.

A. *Software level Encryption*

Here we discuss the various recent proposals among the software encryption and understand why Group Symmetric keying is the best available option.

1) *Voting schemes* [20] requires nodes to be present and vote on the authenticity of the message. Message should gather a minimum number of votes for it to be declared authenticate. Receive history of nodes should match. Since all nodes have to vote, it seems inappropriate for a real time system.
2) *TESLA* [21] is broadcast protocol where there are two messages sent. First being the frame with normal payload and the second is the authentication message. The node has to buffer all the messages until it receives the authentication message. This delay is around 1-10ms but if the delay increases beyond a point then it is not suitable for real time system.
3) *CANAuth* [22] is a protocol that uses ID based key allocation. The problem with this proposal is that the IDs are broadcasted are too many in number and one key for every ID increases load on memory. Keys are broadcasted over the network to respective IDs which exposes the key to an attacker.
4) *MaCAN* [23] employs shared keys between two nodes and CBC-MAC based authentication tags. Details on how the keys should be shared is not mentioned in the proposal. Nodes can also be grouped based on the same trust levels and they can share the same key but there is no formula to decide the trust levels.
5) *LiBrA-CAN* [24] is similar to MaCAN but here the protocols mixes the keys in groups of nodes rather than pairwise. The protocol has a significant more computational load but offers a very good security level when compromised nodes are in minority.
6) *CaCAN* [25] is centralised authentication protocol over CAN where authentication of messages happen inside a centralised node. The central node is expected to have a very high computational power. The problem with this protocol is that if the central node fails then the entire system is compromised.

In the above subsection we looked at all the software protocols and problems on how keys are distributed. It is evident that MaCAN [23] and LiBrA-CAN [24] provide a more better approach to keying than the rest as the number of keys are less and also provide good security. We now revisit all the above protocols from the way the keys are used in the techniques.

1) *Single Authentication keying* is the simplest keying protocol where all frames are authenticated using only one key. All the nodes are in possession of the key and if one node is corrupted, the entire network is corrupted and security is lost. Each frame carries only one tag and thus the increase in bandwidth and computational load is minimal. CANAuth [22] and CaCAN [25] use this keying method. ID oriented keying is a similar variation of this keying method.
2) *Pairwise Keying* is a keying technique where unique authentication keys are shared between each distinct pair of nodes. This procedure is used by MaCAN [23] protocol. The problem in this method is that, if there are *n* nodes and we need to broadcast a message, then *n-1* authentication tags need to be calculated. For higher value of *n* the overhead becomes too big to fit in the frame size of CAN-FD or FlexRay. This procedure can be extended to groups of nodes instead of pairs.
3) *Group Keying* is a successor to pairwise keying where keys are shared between groups of nodes that are on the same trust level or functionality. The size of the group can vary from 2 (pairwise keying) to n (Single authentication keying). The security is not compromised until the corrupted nodes are in minority. This method has been employed by LiBrA-CAN [24] protocol.
4) *TESLA-like keying* requires authentication keys to be broadcasted at regular time intervals. The regular frames are sent over the network first, following which a frame with authentication is sent. The additional overhead here is the buffering of the received frames till the arrival of the authentication key and authentication delay.

Group Keying from all the above keying techniques provide the best trade-off between the security and computational power required for the encryption. We also know that Asymmetric encryption like public-private keys are not possible in automotive networks due to the requirement of higher computational power thus we go for Symmetric keying. Combining Group keying along with symmetric keying provides a very secure environment. The results on the above techniques and why Group Symmetric Keying performance is the best among the recent proposals are explained in Section IV.

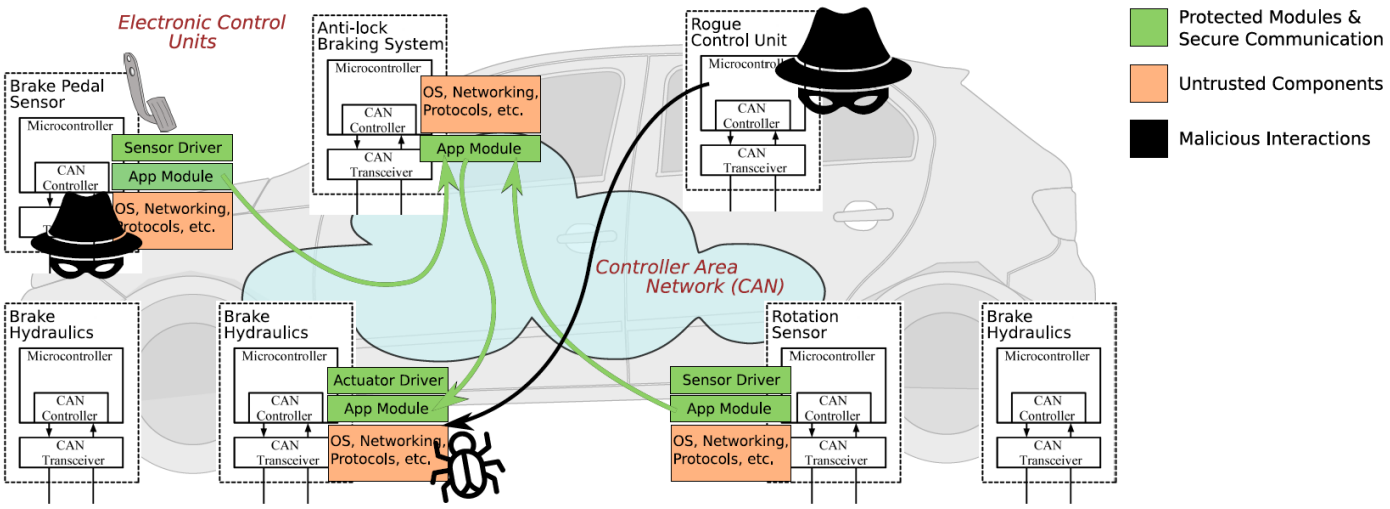

Figure 3. CAN network scenario to illustrate security attacks on CAN network and security guarantees offered by vulCAN

B. vulCAN and secured CAN Components

In this section we look into the brief overview about our second solution vulCAN. We also discuss security offered by vulCAN and an attacker model and how the model is going to test vulCAN solution for the security it offers. vulCAN mainly focuses on compartmentalisation where application is divided into smaller group of trustworthy authentication software components. vulCAN support mutually distrusting components on a single ECU such that code executing on one component cannot adversely affect another component in the CAN network except for the classical denial of service attack. We show the working of the vulCAN by a simple example. In Figure 3, there is an authenticated path that is setup between a software component a braking pedal and wheel rotations to the Anti lock braking system (ABS) all the way to brake hydraulic actuator. vulCAN guarantees that any braking action should come from the braking pedal and not any other module. If a rogue control unit initiates a braking action on the brake hydraulics then the secure app module of the hydraulics checks with the brake pedal sensor if there is a braking pedal actually pressed by the user via the secure path that is created with encryption. If the braking pedal sensor sends a negative response saying that there was no brake pedal pressed then the vulCAN will safely reject the message from the corrupted node and warn the user about the on going attack on the dashboard.

In the following subsection we introduce an *attacker model* which attacks the vulCAN system in order to test it for the security it offers and also mention the *problem statements*, *protocol requirements* of how a secure system is to behave and how the *protocols are addressed by vulCAN*.

1) Attacker Model: An attacker's goal is to impersonate a protected component and send out random messages as though it has been originated from a valid sender component. vulCAN offers protection against two types of attack namely *arbitrary message manipulation* and *arbitrary code execution*.

Arbitrary Message Manipulation. It is a scenario where attacker has gained complete access to the network and can do the following things (i)Broadcast own CAN message with random ID and payload (ii) Observe and record all traffic (iii) destroy or modify any message.

Arbitrary Code Execution. There have been attacks in the past that allows attackers to execute their own code on the ECU which means that the attacker can compromise all software including privileges operating system code and support software except the authenticated software components that are protected by Sancus.

2) Problem Statement: In this subsection we mention the requirements and challenges for a trustworthy component authentication on an unmodified CAN bus. We give details about the expectation from a protocol and system requirements for keeping the network secure. The list of *protocol requirements* that we should expect for providing a secure environment are
P1: Message Authentication. A receiver component should get a strong guarantee that message with specified ID and payload was sent by a trusted sender component.
P2: Lightweight Cryptography. Public key cryptography is ruled out due to higher load since ECUs are constrained with lower computational power and storage space.
P3: Replay Attack Resistance. Packet loss should be predicted and make sure that those packets are not used by attacker again as replay attacks.
P4: Backwards Compatibility. The authentication protocols should incorporate the existing legacy ECUs which do not have any authentication mechanism nor computational power for lightweight cryptography.

Following are the *System level guarantees* that has to be considered while providing complete security
S1: Real-Time Compliance. The authentication scheme should provide fast authentication and meet stringent real-time deadlines when not under attack.
S2: Component Isolation. Integrity of message processing,

authentication algorithms, key confidentiality and management should protected from the attacker.

S3: Component Attestation. The motorist should get a strong guarantee that the critical software components are loaded on ECU and have not been tampered with.

S4: Dynamic Key Update. The system should support secure dynamic key provisioning at run time and also allow broken ECUs to be replaced by a distrusted automobile repair shop.

S5: Secure Legacy ECU Integration. Automotive suppliers cannot be expected to adopt new hardware/software immediately for authentication purposes. It should be possible to shield existing ECUs as a transition measure.

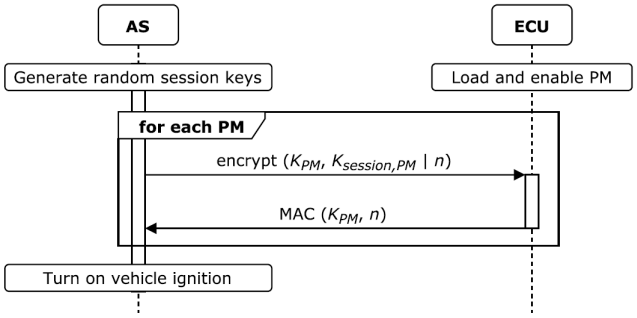

Figure 4. Component isolation and key provisioning by AS

3) Protocol(P1-P4) Requirements Addressed by vulCAN: P1 and P2 requirements are satisfied by associating a symmetric 128-bit cryptographic key with each authenticated CAN identifier. Multiple IDs share the same key to reduce memory consumption costs. MAC is computed using the key, ID and payload including a monotonically increasing counter to protect against replay attacks(P3). P3 requirement demands that replaying a previously authenticated message should result in that message being discarded. For this we use a nonce as a source of freshness. Identical nonce values should never be reused under the same key and associated data. A common way to address the nonce initialisation is the use of short term session keys. At system boot or whenever counter overflows, both sender and receiver generate a fresh session key based on a larger epoch counter. Due to loss of packets in the network the nonce values need to be re-synchronised. If MACs get out of sync then message authentication will fail continuously until the system is re-synchronised. Vulcanised LeiA [16] uses a small 16-bit nonces in the extended CAN identifier field and re-establishes session keys on the counter overflow. To be able to recover from a long term network failure, a receiver can send a error frame that prompts a sender to broadcast its current epoch counter value. LeiA [16] has chances of breaking the legacy application that relies only on availability of extended CAN identifiers. Another solution to deal with packet loss that is provided in vatiCAN [15] is the use of larger 32-bit nonce which relies on a trusted global Nonce Generator (NG) component to periodically reset nonces in the entire network. Every few milliseconds, NG broadcasts a randomly chosen value to be used by all participating ECUs as the new value for all counters.

4) System Requirements met by vulCAN: Recall that the payload of a authenticated CAN message is sent first, followed by the authentication frame containing the MAC. VatiCAN [15] makes it more efficient by parallely computing the MACs. The sender first sends the message m and starts to compute the MAC. The receiver receives the message m and starts to compute the MAC till then the sender sends the authentication frame thus satisfying the requirement of S1 dealing with Efficient MAC computation. The vulCAN using Sancus's hardware level authenticated encryption primitive computes MAC faster. Due to CAN limitations we truncate the resulting MAC by discarding the eight least significant bytes. Due to hardware-level cryptography the MAC computation time decreases with an order of magnitude(S1).

Software Isolation and Key storage(S2) - Security of a authentication scheme relies the key management. The authenticated CAN messages are constructed and verified inside the Sancus such that the connection keys do not leave the private data sections. This also protects the critical application software which reacts to authenticated messages thus providing untampered execution (S2).

Software Attestation and Key Provisioning(S3, S4) - All guarantees are maintained only after PM authenticated components are completely isolated and provisioned with symmetric cryptographic keys. When a system is rebooted an attacker has full control until the car is turned on. This leads us to three challenges (i) attest the integrity of critical distributed software components, (ii) establish session keys over untrusted CAN bus, (iii) replace broken ECUs in an untrustworthy automobile repair shop.

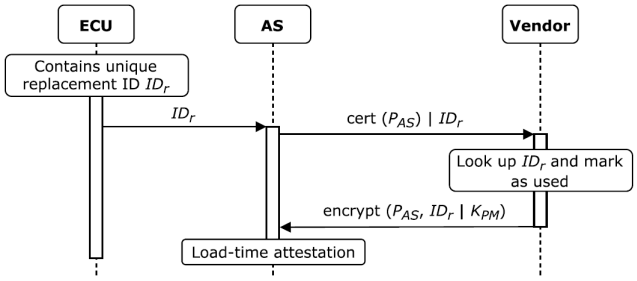

Figure 5. ECU replacement method using AS

These challenges are overcome by introducing an Attestation Server (AS). The AS possess Sancus module specific 128-bit key K_{PM} of every authenticated software component in the network. When starting the vehicle a simple challenge response attestation happens between AS and each module. The AS sends a encrypted challenge to every module and waits for a response as shown in Fig. 4. Once the response is obtained, it assures the complete attestation and isolation of that module (S3, S4) and also satisfies the first challenge. The second challenge is solved by using dynamic key provisioning by sending an encrypted challenge containing the new key to existing module. The module then decrypts the message and obtains the new key to be used for future authentication. Addressing the final challenge, ECU replacement, we

replace the AS with asymmetric cryptography. When a ECU joins a particular network, it broadcasts a unique replacement identifier ID_r. Upon receiving ID_r, AS sends its public key and ID_r to remote car vendor over an untrusted network. The car vendor acts as a trusted infrastructure provider in the Sancus key hierarchy. Upon receiving ID_r and AS ceritifcate cert(P_{AS}), the vendor computes the module specific K_{PM} and uses the AS's public key to send them securely back to the AS (Fig. 5). Thus AS can now proceed with load-time attestation.

Shielding Legacy ECUs (S5) - In order to make old ECUs compatible with vulCAN, we place Sancus enabled gateway in front of a legacy ECU and perform authentication on behalf of the shielded legacy components. The gateway holds the message until authentication happens and then if message is authentic then it passes on the message over the trusted private CAN bus between the gateway and legacy ECU.

Security Analysis of vulCAN - The session keys are 128-bit long hence it would take a long time to brute force all 2^{128}keys. Attacker can try correctly guessing the MAC which is 64 bit long but the probability is 2^{-63}. The attacker has a very short time to guess the keys since session keys keep changing. We trust the processor, authenticated CAN components, authenticated softwares, global attestation server.Adversary model where the attacker damages or extracts the key physically is out-of-scope. Since the adversary does not get any key or access to any protected module, the only leverage left is to launch a denial-of-service attack by claiming all replacement IDs. Such an issue can be dealt with at the application level. By the above overview of vulCAN we see that it provides a security for various types of attacks and also meets the real time deadlines without compromising on security.

IV. Experimental Evaluation

A. *Software Keying and results*

1) Network Setup: In this section we discuss the network setup that is going to be used for testing of the Software proposals and the performance level of each software keying techniques which is the first solution after which the second solution vulCAN experimental results are evaluated.

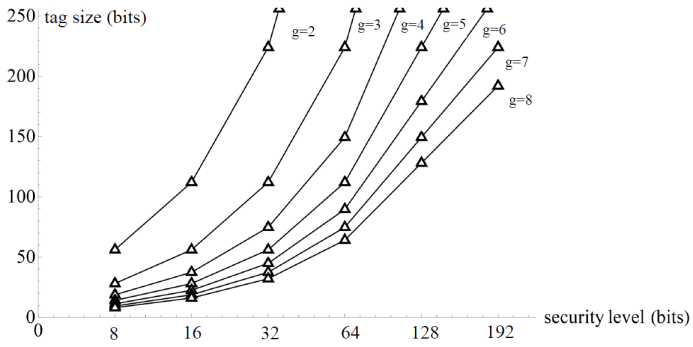

Figure 6. Tag size for a security level of 8 to 128 bits with n = 8

Consider a scenario where there are 8 nodes i.e. n = 8 and we group them in different group size g. If g = n then it is *single authentication keying* where there is only one key for the entire network. If the group size is g = 2 then it corresponds to *pairwise keying* which results in 7 authentication tags for a broadcast message. For a security level of 128 bits the tag size grows almost up to 896 bits which crosses the payload size of CAN-FD and FlexRay. Finally we see that group size of 4 or 5 seems to have a nice trade-off between security and tag size as shown in Figure 6 which is called *group keying*.

For a group size g = 8, if one node is corrupted then all security is compromised since there is only one key for the entire network. For a group size g = 2, the security is highest since the corrupted node affects only one node and the rest remains secure even if one node is corrupted. The trade off comes well in group 4 and 5 where the security is not compromised until the corrupted nodes are in minority.

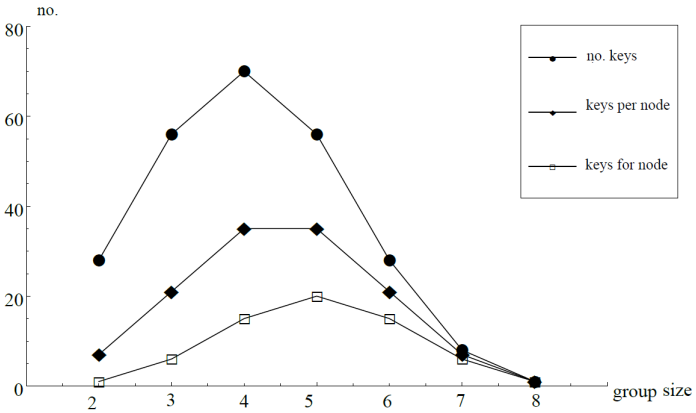

Figure 7. Comparison on the number of keys, number of computed tags and number of verified tags with n = 8 and g = 2,3..8

Figure 7 shows the comparison between the number of keys, keys per node and key for receiving nodes. The highest number of keys are for a group of 4 and least for a group of 8 as it has only 1 key. Even though the number of keys are highest for g = 4, in later results we see that the number of authentication tags to be computed are less compared to group size g = 2.

The network for the experiment were designed to comply with real world in-vehicle networks used by automotive industry. The experimental setup have 30 ECUs which are into 5 clusters such that each of them have similar load of around 2200 frames/second/ECU. The number of ECUs was chosen to represent the maximum number of ECUs specified in ISO 11898 for maximum signalling rate of 1Mbps. The same topology is used for both FlexRay and CAN-FD.

The comparative evaluation of the protocols can account for two distinct characteristics : the *bandwidth* (number and size of authentication tags) and the *computational load* on each ECU (depends on the number of tags that each ECU has to compute). The bus simulation happens over a simulator called CANoe. The computational load is to be neglected

as hardware implementations are available for cryptographic primitives e.g., CBC-MAC can be straight forwardly derived. Hardware implementation will allow computation is several micro-seconds whereas high grade ECU will cost dozens of micro-seconds for the same cryptographic computation.

We briefly discuss the results of each the keying technique.

1) *Single Authentication Key* implies that only one tag added to each frame. The maximum increase in payload is 128 bit for a security level of 128 bits which is the minimum compared to other keying techniques. The computational load of a receiving ECU increases proportionally with number of sent/received frames per second.
2) *Pairwise Keying* requires the number of authentication tags to be equal to the number of ECUs that are receiving the frame. The worst cast scenario is when an ECU has to broadcast a message. Our experimental setup has 30 ECUs, which translates to 29 authentication tags/frame. The size of overhead for 64 bits and 128 bits security level is 232 and 464 bytes which exceeds the payload size of CAN-FD (64 bytes) and FlexRay (256 bytes). Therefore this approach is not feasible in terms of computational load.
3) *TESLA like keying* requires for each sender a new frame containing 32, 64, 128 bit authentication tag that is periodically broadcasted. As the delay between the message and authentication message increases the ECU has to buffer more and more messages thus there is an exponential increase in memory consumption which exceeds the memory of an ECU which is rather small.

Authentication protocol	Tag size [bits]	Max. payload [bits]	Max. payload [bytes]
Single authentication key	32	32	4
	64	64	8
	128	128	16
Pairwise keying	32	928	116
	64	1856	232
	128	3712	464
TESLA-like keying	32	32	4
	64	64	8
	128	128	16
Group keying	32	192	24
	64	384	48
	128	768	96

Table 1. Number of tags added to a frame when applying group keying

4) *Group Keying* is a concept that provides more realistic deployment method by reducing the number of keys. For our experimental network, by grouping 30 ECUs into 5 clusters, the payload increase is considerably smaller than pairwise keying. The increase in 5 clusters can be viewed as 5 ECUs that can be grouped in pairs of 2,3 or 4. The maximum number of authentication tags added to a frame will be the number of shared keys between the sending cluster and the receiving clusters. This is summarised in Table 1. The maximum number of authentication tag added is 6 to each frame. Group keying can be successfully simulated on our experimental network using 32, 64 bit authentication tags. The multiple authentication tags lead to an increase in computational load where one ECU has to compute 6 tags but this can be further mitigated by hardware implementation. The additional busload increased is not much significant. Table 1 indicated that by gathering clusters with nodes that share the same key brings the optimal trade off between security, computational load and payload.

Table 2 indicates the maximum payload increase in each of the cases. Pairwise keying has the highest increase in overhead which cannot fit inside the frame of CAN-FD and FlexRay since it crosses the payload capacity of the protocols. Group keying has optimal compromise in terms of the overhead which in between that of single authentication keying and pairwise keying.

Authentication protocol	Tag size [bits]	Max. payload [bits]	Max. payload [bytes]
Single authentication key	32	32	4
	64	64	8
	128	128	16
Pairwise keying	32	928	116
	64	1856	232
	128	3712	464
TESLA-like keying	32	32	4
	64	64	8
	128	128	16
Group keying	32	192	24
	64	384	48
	128	768	96

Table 2. Payload for authentication protocols on all tag sizes

The computational load increase is highest for group keying which exceeds that of TESLA like keying and Single Authentication Keying (Table 3). This can be reduced by hardware implementation of the cryptographic MAC computation.

Authentication protocol	MIN comp. load [tags/s]	MAX comp. load [tags/s]	AVG comp. load [tags/s]
Single authentication key	1237.5	3290.625	2127.8125
TESLA-like keying	1237.5	3290.625	2127.8125
Group keying	3712.5	9871.875	6393.4375

Table 3. Computational load for the considered authentication protocols

Finally, TESLA and group keying stay close to the busload performance of the baseline CAN network. The more efficient Single Authentication key has only 10 - 20 percent advantage in the bus load which is unclear if that is worthy given the fragile security it offers as shown in Table 4. We can also observe that, for the FlexRay network, the busload changes only when additional frames are introduced, the addition of data bytes to the existing frames has little effect. This is due to global payload size is set while configuring the network. In CAN-FD, adjustment of each frame's size is possible whereas in FlexRay the frame size is configured to global payload size

even if the payload does not contain any data. Thus by the experimental results we can see that the Group Symmetric keying is by far the best among the software keying proposals and is easier to implement.

Authentication protocol	Recorded busload: CAN-FD [%]	Recorded busload: FlexRay [%]
Baseline	43.36	58.55
Single authentication key	48.97	58.57
TESLA-like keying 10 ms	64.97	71.61
TESLA-like keying 20 ms	57.40	65.89
TESLA-like keying 40 ms	53.62	61.56
TESLA-like keying 80 ms	51.73	59.41
Group keying (groups of 2)	68	58.57
Group keying (groups of 3)	82.21	58.57
Group keying (groups of 4)	70.51	58.57

Table 4.Recorded busload on CAN-FD and FlexRay

B. *vulCAN evaluation and results*

Here we discuss the experimental evaluation of the second solution vulCAN. All experiments were conducted featuring six Xilinx Spartan-6 FPGAs Sancus enabled OpenMSP430 core[13,33] running at 20MHz. Each Sancus core was configured to provide 128 bits security. ECUs were interfaced over SPI with MCP2515 CAN transceiver chips with speed of 500kbps. All source code were compiled with Sancus C compiler based LLVM/Clang v4.0.0 with optimization set for size (-0s). The experimental evalution will be discussed on basis of memory consumption, performance of the system.

1) TCB size and Memory footprint: The vulcanized vatiCAN [15]/LeiA [16] was implemented with a small C library that leverages Sancus hardware level authenticated encryption primitives. When the library is included in a PM, the remaining software stack remains untrusted regarding MAC computation integrity and confidentiality of key material. Trusted Computing Base (TCB) encompasses only 22 lines of trusted source code for LeiA [16] and 147 lines for vatiCAN [15]. Regarding the unprotected stack, CAN driver alone requires 322 lines of code against the 670 lines of code for the popular CAN bus shield for Arduino devices. The memory footprint of unprotected CAN driver measures 2482 bytes. For the vulcanized vatiCAN [15]/LeiA [16] libraries on the other hand has a total binary size of 790/1818 bytes when compiled as an unprotected application and 906/1948 bytes when computed as part of a protected module. The slight increase is due to the compiler generated code stubs inserted on every call to and from the unprotected CAN driver. VatiCAN [15] further requires 22 bytes of metadata for each authenticated connection and LeiA [16] requires 44 bytes due the ID, epoch counter and long/short term 128 bit keys.

2) Performance Evaluation: For a fair overhead evaluation, vatiCAN [15] was evaluated on a 16 MHz ATmega 8-bit ECU.

MAC Computation - For every secure message, vatiCAN [15] computes a 64 bit MAC over 112 bits of associated data (ID, payload, nonce). This operation requires 47600 clock cycles/ 2.95 ms. The hardware assisted encryption requires 4222 clock cycles or 0.21 ms to compute the 128 bit MAC. Since vulCAN uses only 16 bit nonces, which reduces the associated data to 96 bits and MAC computation time down to 4049 clock cycles / 0.20ms . Cryptographic acceleration is crucial to fulfil real-world automotive safety requirements. At a highway speed of 100kmph, vatiCAN software based MAC computation corresponds to a distance of 8cm, whereas vulCAN reduces the distance to 0.6cm thus achieving the results faster and meeting real time deadlines much sooner.

Scenario	Cycles	Time	Overhead
Legacy (standard ID)	8,135	0.41 ms	–
Legacy (extended ID)	9,620	0.48 ms	18%
vatiCAN (extrapolated†)	58,948	2.95 ms	625%
Sancus+vatiCAN (unprotected)	15,570	0.78 ms	91%
Sancus+vatiCAN (protected)	16,036	0.80 ms	97%
Sancus+LeiA (unprotected)	18,770	0.94 ms	131%
Sancus+LeiA (protected)	19,211	0.96 ms	136%

Table 5.Overhead to send an authenticated CAN message with/without Sancus encryption and software protection.

Message Transmission - The overall overhead to send an authenticated message is mentioned in Table 5. The first row shows the time required by our CAN driver to send a plain unauthenticated message with an 11-bit standard ID (about 0.41ms). vulCAN must send two messages for every legacy message to be authenticated. The baseline measurement in Table 5 also includes the time needed by the transceiver chip to initiate transmission and signal acknowledgement from receiver device. vulCAN starts the MAC computation when it receives the CAN message and after the arrival of authentication message from the sender can it authenticate the computed MAC. This optimization technique reduces the over the overall time to send one message lesser than sending two unauthenticated legacy messages. Comparing the third and fourth rows of Table 5 reveals that the total relative overhead for sending an authenticated message with Sancus+vatiCAN decreases the relative overhead with a huge 534 percent, as compared to the extrapolated software only vatiCAN [15]. Second row of Table 5 reveals that sending a legacy CAN message with an extended 29 bit identifier is slightly more expensive, as extra device registers must be loaded. Compared to the Sancus+vatiCAN/LeiA unprotected, the protected case has a smaller increase in overhead from 5 to 6 percent due the transparently copying of message memory buffers to and from unprotected CAN driver.

Round Trip Time - To assess authentication delays in a sender/receiver scenario, we look into the round trip exper-

iment shown in Figure 8. The performance overhead is expected to be around 100 percent. Comparing table 5 and table 6, the relative overhead for vulcanized vatiCAN [15]/LeiA [16] further decreased, down to 74 and 112 percent.

Scenario	Cycles	Time	Overhead
Legacy	20,250	1.01 ms	–
vatiCAN (extrapolated†)	121,992	6.10 ms	502%
Sancus+vatiCAN unprotected	35,236	1.76 ms	74%
Sancus+vatiCAN protected	36,375	1.82 ms	80%
Sancus+LeiA unprotected	42,929	2.15 ms	112%
Sancus+LeiA protected	43,624	2.18 ms	115%

Table 6 Round-trip time intervals protection.

This shows the fact that messages also have to be received and processed by the receiver, making the contribution of MAC computation less important in the overall timing. Induced Sancus software protection overhead adds only 3 to 6 percent and can be reduced further if the protected modules spend more time processing the message. The software only vatiCAN, demonstrates once more that MAC calculation remains dominant, increasing the relative overhead with over 400 percent. The above overheads may be different for some real world applications due to restrictions by other softwares in the system, which is beyond the scope of this paper. In comparison with state-of-the-art software-only solutions, the results show that vulCAN can target substantially larger class of safety critical applications where real-time deadlines must be satisfied.

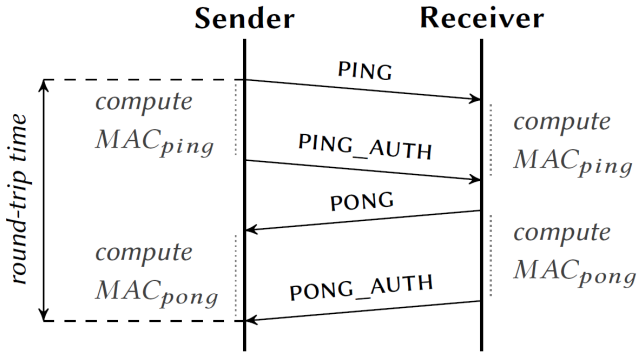

Figure 8 Round-trip time experiment timing overview.

Extended Application Scenario of vulCAN - Figure 9 shows the photograph of the vulCAN setup, which consists of two off-the-shelf automobile instrument clusters from the 2014 Seat Ibiza model and six Sancus-enabled ECUs that we configured as described in experimentation evaluation. Further, CAN-to-USB converter is attached to be able to monitor traffic and inject CAN messages from an ordinary laptop computer. Fig 10 provides a simplified schematic of the Fig 9. The demo schematic involves a central ECU that has a protected Sancus software module PM_{CS} to securely process commands from legitimate motorist. The keypad device is used as an abstract of motorist to accelerate, brake etc. To gain exclusive access to keypad peripheral, PM_{CS} securely links to a local PM_{key} MMIO driver assembly module. For demonstration, there is another red colored keypad which is unprotected.

Figure 9 Application scenario with original instrument clusters and Sancus-enabled ECUs

The demo scenario has 4 wheel ECUs for actuating brake, LED display for wheel RPM. The two instrument clusters represent unmodified legacy devices, where one is protected using Sancus enabled gateway and the other is unprotected. The gateway ECU hosts a PM_{gw} component that listens to selected instrument cluster IDs, making sure that only authenticated messages are forwarded. When key is pressed on the black keypad, the distributed wheel ECUs respond to authenticated messages and update their wheel RPM. When a key is pressed on the attacker keypad, the unprotected instrument cluster shows speed changes while the protected Speedometer remains unchanged and notify the user about the on going attack. vulCAN guarantees that any critical actuator even can only be caused by a chain of processing events that an ultimately be traced back to there authenticated component PM_{key} that originally sensed the keypad input provided by the motorist.

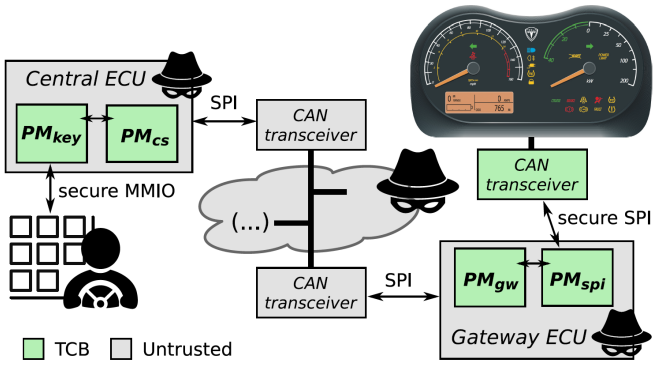

Figure 10 Schematic of the demo scenario depicted in Fig. 8

V. Related Work

This section compares the vulCAN and Software keying proposals. LiBrA-CAN [24] which uses group symmetric keying when tested on a high end ECUs and laptop CPUs shows results similar to that of vulCAN. But vulCAN provides similar authenticity while being AUTOSAR-compliant and relies on much lighter ECUs and off the shelf CAN hardware. All the software proposals approaches discussed require some form of specialised hardware to efficiently implement cryptography. They also do not address challenges of key distribution, software integrity and key confidentiality in the presence of active ECU attackers. The vulCAN on other hand guarantees security and implements light weight trusted computing technology.

Sancus has been applied in secure smart metering infrastructure where multiple sensors are attached to assets like water pipes, HVACs and electric meters to detect problems and usage of resources. These can be used to for predictive maintenance and can solve problems even before they occur. Sancus allows PM to securely inspect and attest a host operating system on a lightweight computing node. In automotive industry, this technology could be used to integrate third-party hardware or software components into vehicle allowing the manufacturer to securely monitor their behaviour. These variety of features of vulCAN are not available in software keying techniques which plainly works on message encryption and decryption without focusing on subtler aspects of key management and component security.

VI. Conclusion and future work

The conclusion for the two solutions can be drawn from the experimental section. For the Software keying proposals, Pairwise keying leads to a minimum computational load but higher number of authentication tags. Single Authentication keying offers good one tag one key which has a very low computational load but lacks security. TESLA like keying depends on the time interval between normal CAN message and authentication message thus with the increase in time interval the memory consumption increases exponentially. Thus *Group keying* being the solution where there is a good trade off between number of authentication tags and security. *vulCAN* a trusted computing design for message authentication plus software component attestation and isolation in vehicular communication network contributes significantly in the future technology of autonomous driving with its advanced security features. vulCAN offers results are promising and the solution shows that relatively inexpensive micro-controllers equipped with lightweight embedded cryptography and software component isolation enable strong security guarantees, while maintaining real time deadlines. The choice of authentication, communication technology and trusted computing platform is application specific. If the manufacturer needs a easier, cheap and medium level of security for its automotive network then Group keying is a better choice. With advancement in cars and expensive cars being in market, vulCAN is a better solution due to advance security protections it offers but it is less backward compatible compared to Group keying Technique. VulCAN solution implementation requires more effort and is expensive but provides a very high security compared to group keying. Thus while choosing the protocol manufacturer should decide the protocol based on the cost, effort and time to implement the solution in the product. In future work, the investigation of real-time responsiveness and availibility on partially compromised components can be addressed. Further, impementation of secure vehicular attestation server and investigate application scenario interms oof V2X can be looked into. The Software keying and vulCAN can be used to applying the hardware nodes and actuators in an IoT network and provide a secure path between the Edge gateway and the sensor/actuators.

References

[1] P. Vasile, B. Groza, and S. Murvay, "Performance analysis of broadcast authentication protocols on can-fd and flexray," in *WESS15, October 04-09, 2015, Amsterdam, Netherlands*. Amsterdam, Netherlands: ACM, 2015.

[2] J. V. Bulc, J. T. Mhlberg, and F. Piessens, "Vulcan: Efficient component authentication and software isolation for automotive control networks," in *Proceedings of 33th Annual Computer Security Applications Conference (ACSAC17). ACM, New York, NY, USA*. New York, NY, USA: ACM, 2017.

[3] 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 *USENIX Security 2011*, San Francisco, CL, USA, 2011.

[4] M. Cheah, S. A. Shaikh, O. Haas, and A. Ruddle, "Towards a systematic security evaluation of the automotive bluetooth interface," in *Vehicular Communications 9 (2017), 818.*, 2017.

[5] T. Hoppe, S. Kiltz, and J. Dittmann, "Security threats to automotive can networks practical examples and selected short-term countermeasures," in *In Computer Safety, Reliability, and Security (SAFECOMP 08)*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008.

[6] K. Koscher, A. Czeskis, F. Roesner, a. T. K. Shwetak Patel, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, and et al, "Experimental security analysis of a modern automobile," in *In Security and Privacy, 2010 IEEE Symposium on. IEEE, 447462*. IEEE, 2010.

[7] C. Miller and C. Valasek, "Remote exploitation of an unaltered passenger vehicle," in *Black Hat USA .(2015)*. Black Hat USA, 1966.

[8] ——, "A survey of remote automotive attack surfaces birthday problem," in *Black Hat USA (2014)*. Black Hat USA, 2014.

[9] MarkoWolf, AndrWeimerskirch, and C. Paar, "Security in automotive bus systems," in *In Workshop on Embedded Security in Cars*, 2004.

[10] S. Checkoway, L. Davi, A. Dmitrienko, A.-R. Sadeghi, H. Shacham, and M. Winandy, "Return-oriented programming without returns," in *In Proceedings of the 17th ACM Conference on Computer and Communications Security (CCS 10)*. New York, NY, USA: ACM, 2010.

[11] J. Noorman, J. V. Bulck, J. T. Mhlberg, F. Piessens, P. Maene, B. Preneel, I. Verbauwhede, J. Gtzfried, T. Mller, , and F. Freiling, "Sancus 2.0: A low-cost security architecture for iot devices," in *ACM Transactions on Privacy and Security (TOPS) 20 (2017) 7:17:33. Issue 3*. ACM, 2017.

[12] P. Maene, J. Gtzfried, R. de Clercq, T. Mller, F. Freiling, and I. Verbauwhede, "Hardware-based trusted computing architectures for isolation and attestation," in *IEEE Trans. Comput. 99 (2017)*. IEEE, 2017.

[13] O. Girard, "openmsp430 a synthesizable 16bit microcontroller core written in verilog," in *https://opencores.org/project,openmsp430. (2009)*, 2009.

[14] A. S. . 2016, "Specification of module secure onboard communication," in *https://www.autosar.org/standards/classic-platform/release-43/software-architecture/safety-and-security/*, 2016.

[15] S. Nrnberger and C. Rossow, "vatican vetted, authenticated can bus. in cryptographic hardware and embedded systems," in *CHES 16: 18th International Conference, Santa Barbara, CA, USA, August 17-19, 2016 Proceedings*. Berlin, Heidelberg: Springer, 2016.

[16] A.-I. Radu and F. D. Garcia, "Leia: A lightweight authentication protocol for can," in *In Computer Security ESORICS 16: 21st European Symposium on Research in Computer Security, Heraklion, Greece, September 26-30, 2016 Proceedings*. Springer International Publishing, 2016.
[17] F. Consortium, "Flexray," in *FlexRay Communications System, Electrical Physical Layer Specification, Version 2.1*, 2015.
[18] R. B. GmbH, "Can with flexible data-rate version 1.0," 2012.
[19] B. Groza, P. Murvay, and Ecient, "Protocols for secure broadcast in controller area networks," in *Industrial Informatics, IEEE Transactions on, 2012*. IEEE, 2012.
[20] C. Szilagyi and P. Koopman, "Flexible multicast authentication for time-triggered embedded control network applications," in *In Dependable Systems Networks, 2009. DSN'09. , IEEE/IFIP International Conference on, pages 165174. IEEE, 2009.* IEEE, 2009.
[21] A. Perrig, R. Canetti, J. Tygar, and D. X. S. Ecient, "authentication and signing of multicast streams over lossy channels," in *IEEE Symposium on Security and Privacy*. IEEE, 2000.
[22] A. V. Herrewege, D. Singelee, and I. Verbauwhede, "Canauth-a simple, backward compatible broadcast authentication protocol for can bus," in *In ECRYPT Workshop on Lightweight Cryptography 2011*, 2011.
[23] O. Hartkopp, C. Reuber, and R. Schilling, "Macan-message authenticated can," in *In 10th Int. Conf. on Embedded Security in Cars (ESCAR 2012)*, 2012.
[24] B. Groza, S. Murvay, A. V. Herrewege, and I. Verbauwhede, "Libracan: a lightweight broadcast authentication protocol for controller area networks," in *In Cryptology and Network Security, pages 185 200. Springer, 2012*, 2012.
[25] R. Kurachi, Y. Matsubara, H. Takada, N. Adachi, Y. Miyashita, and S. Horihata, "Cacan - centralized authentication system in can (controller area network)," in *In 14th Int. Conf. on Embedded Security in Cars (ESCAR 2014)*, 2014.