Security shortcomings and countermeasures for the SAE J1939 commercial vehicle bus protocol

Pal-Stefan Murvay, Bogdan Groza

***Abstract*—In the recent years, countless security concerns related to automotive systems were revealed either by academic research or real life attacks. While current attention was largely focused on passenger cars, due to their ubiquity, the reported bus-related vulnerabilities are applicable to all industry sectors where the same bus technology is deployed, i.e., the CAN bus. The SAE J1939 specification extends and standardizes the use of CAN to commercial vehicles where security plays an even higher role. In contrast to empirical results that attest such vulnerabilities in commercial vehicles by practical experiments, here we determine that existing shortcomings in the SAE J1939 specifications open road to several new attacks, e.g., impersonation, DoS, DDoS, etc. Taking advantage of an industry-standard CANoe based simulation, we demonstrate attacks with potential safety critical effects that are mounted while still conforming to the SAE J1939 standard specification. We discuss countermeasures and security enhancements by including message authentication mechanisms. Finally, we evaluate and discuss the impact of employing these mechanisms on the overall network communication.**

***Index Terms*—J1939, commercial vehicles, security, authentication.**

I. Introduction

VEHICLES incorporate a multitude of systems which work together with the goal of making their usage an easier and safer experience. One important step in the evolution of the automotive industry was the adoption of bus systems for organizing module interconnections as an in-vehicle network. Various network technologies such as the Local Interconnect Network (LIN), Controller Area Network (CAN), FlexRay and more recently 100Base-T1 Ethernet were designed to cope with the requirements given by the increasing traffic requirements. However, CAN is still the most commonly used network in automotive applications as it provides suitable performance and acceptable cost for the majority of modules. More recently, CAN-FD (CAN with Flexible Data-Rate) was created as an improvement of CAN that allows bigger payloads (64 bytes data field instead of 8) and higher transmission speeds for the data field.

All of the commonly used protocols for in-vehicle communication share the same lack of support for security functionalities. As the security of vehicular networks became an intensely studied topic, car security was substantially explored in the past few years. Extensive experimental analysis done by various research groups, e.g. Koscher et al. [1], Checkoway et al. [2] and later Miller and Valasek [3], decisively proved the lack of proper security mechanisms on passenger cars. As expected, since CAN is the most employed communication protocol, the majority of the reported attacks are using CAN as an entry point.

Extending the aforementioned context, our work focuses on the commercial vehicle sector and, in particular, on the security shortcomings of the J1939 protocol, a higher layer protocol which uses CAN at the physical and data-link layers. We first discuss recently reported shortcomings of J1939 that were used to mount various types of denial of service (DoS) attacks.

A. Recently reported attacks on J1939

While most of the research efforts so far were focused on passenger cars, it was only recently that the commercial vehicle sector come to attention.

Dariz et al. [4] discuss general security aspects of higher level CAN-based protocols employed in heavy duty vehicles (eg. J1939 and ISOBUS) and the outlook of migrating to Ethernet-based in-vehicle networks. Burakova et al. present their findings in analyzing the security of SAE J1939 networks on commercial vehicles [5]. Their research was focused on inject and replay attacks that were already reported for consumer vehicles. This category of attacks is common to all CAN based networks regardless of the higher layer applications. Specific J1939 protocol vulnerabilities were not considered in their work.

Mukherjee et al. take the first steps into determining the exploitable features of the J1939 protocol [6] and present three DoS attacks. Only two of these attacks are validated on actual commercial vehicle modules. The first one consists in sending a large number of requests to a target node in an attempt to increase its computational load to a point in which it cannot fulfil the timing requirements of cyclic message transmission. The second attack targets the usability of the J1939 transport protocol (employed for sending multi-frame messages) preventing further requests for such messages from being answered by establishing never ending connections.

Their third attack is also aimed at multi frame messages sent by the J1939 transport protocol, but is mounted only on self built nodes not on the real-world modules of their setup. However, this attack may be based on a wrong assumption, i.e., that once a node establishes a connection with a second, a third node could intervene misleading the second node to allocate a smaller receive buffer which will yield and overflow when the first node sends the larger amount of data. However, according to standard specifications in Chapter 5.10.3.2 from

Pal-Stefan Murvay and Bogdan Groza are with the Department of Automatics and Applied Informatics, Politehnica University of Timisoara, Romania, e-mail: {pal-stefan.murvay, bogdan.groza}@aut.upt.ro

[7] the first connection should be aborted by the legit initiator node as a result of the clear-to-send message sent for the second connection request and thus no buffer overflow should occur. In one of the attacks that we discuss, we specifically use clear-to-send messages to abort connections. Moreover, even if the initiator node does not abort the connection due to a faulty protocol implementation, the receiver node should be able to detect the faulty behaviour based on the packet sequence numbers which would be out of order or exceeding the maximum number of packets declared in the second connection request.

B. Related work on CAN bus security

The subject of securing CAN-based communication has drawn lots of attention following the multitude of attack reports. As a result, various approaches for providing authenticated communication in CAN networks were proposed.

The MaCAN authentication protocol proposed in [8] uses an authentication scheme based on message authentication codes (MACs), the protocol was later found to be flawed by formal verification [9]. Similarly, Vecure [10] and LeiA [11] rely on symmetrically shared keys and MACs for data authentication. Groza et al. studied trade-offs for a TESLA-based authenication protocol [12]. LiBrA-CAN provides authentication based on key sharing in groups of nodes [13]. More recently, group key-sharing is addressed by Jain and Guajardo [14] with keys that are securely exchanged by exploiting physical properties of the bus, a technique that was initially proposed by Mueller and Lothspeich [15]. Kurachi et al. propose the use of additional hardware mechanisms for discarding malicious frames that do not pass the authentication test [16]. A security-aware mechanism for signal allocation on CAN frames is discussed by Lin et al. in [17] and [18].

With the introduction of CAN-FD, more efficient transmission of bigger payloads is possible, paving the way for new security mechanisms. Woo et al. propose a security architecture that can provide confidentiality, authentication, access control and key management for CAN-FD communication [19]. A previous report from the same authors discusses attacks and countermeasures on the CAN bus [20]. We also take advantage of CAN-FD and propose an authentication mechanism to mitigate the exploitable weaknesses of the J1939 protocol.

The rest of this paper is structured as follows. Section 2 sets the background by introducing technical details of the CAN and SAE J1939 protocols. We continue with a security analysis of the SAE J1939 protocol specification and present the attacks that we identified in section 3. In section 4 we propose countermeasures for the identified vulnerabilities by using authentication mechanisms. The feasibility of the presented attacks and the performance impact of the introduced authentication mechanisms are evaluated in section 5 using a J1939 network simulation.

II. Background

We first give some context information on CAN and the SAE J1939 specification.

A. Controller Area Network

The CAN [21] protocol specification, initially developed by BOSCH, has been widely adopted by the automotive industry. Newer versions of the CAN protocol developed for various physical layers are defined in the ISO 11898 standard family for which ISO 11898-1 [22] describes the general architecture of CAN. Most commonly, CAN is used as a two wire differential bus that can support baudrates of up to 1Mbit/s.

CAN was designed to provide broadcast transmission with collision avoidance features but it offers no security-related mechanisms. The access to the CAN bus is priority-based depending on the message identifier and is decided in the arbitration phase of the message transmission. The lower the value of the CAN identifier, the higher the priority. The CAN frame, tailored to achieve these features, consists of the following main fields: start of frame, arbitration field (which includes the message identifier), control field (which includes payload length), data field up to 8 bytes in size, CRC, acknowledgement field and end of frame. In its initial version, the CAN specification accounted for standard frames with 11-bit identifiers. Starting with the CAN 2.0 specification Part B extended frames were introduced which use 29-bit identifiers by adding 18 bits in the arbitration field. Figure 1 illustrates a simplified representation of an extended CAN frame with an emphasis on the arbitration field structure.

Standard and extended frames can coexist on the same CAN bus as long as its implementation complies with version 2.0 or newer of the CAN specification.

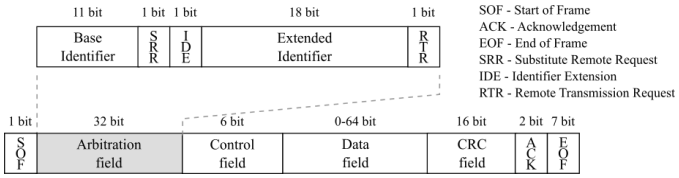

Fig. 1. Simplified representation of an extended CAN frame

The increasing bandwidth demands of the automotive industry lead to the development of CAN-FD which improves on the shortcomings of CAN by offering a higher data field of up to 64 bytes in size and bit rates for the data field of up to 8Mbit/s.

The CAN specification only describes the physical and data link layer allowing the usage of application-specific higher layer protocols. The industry is relying mostly on manufacturer-specific upper-layer protocols which are not disclosed to the public in an attempt to increase security through obscurity. Only a handful of higher layer protocols based on CAN are standardised, some of which are widely used in all sectors of the automotive industry such as the diagnostic protocols, e.g., Unified Diagnostic Services (UDS). Other protocols target specific industry areas such as commercial vehicles for which the Society of Automotive Engineers (SAE) defined the J1939 protocol.

B. SAE J1939

Even though the SAE J1939 standard collection also contains definitions for the lower layers of the protocol (physical and data link) the protocol is basically built on the CAN layers with the introduction of additional rules that are implemented in software. SAE J1939 specifies baudrates of 250Kbit/s and 500Kbit/s and the use of extended 29-bit CAN identifiers. The presence of standard frames is allowed on the bus but SAE J1939 messages are only sent as extended CAN frames.

The SAE J1939 frame format and transport protocol are presented in the J1939-21 document [7]. The SAE J1939 frame format breaks down the identifier field into three main components: *priority*, *Parameter Group Number (PGN)* and *source address* as shown in Figure 2. PGNs are values that uniquely identify a parameter group which encapsulates a set of signals with a common context. This grouping of signals related to a common process and sharing the same transmission rate is destined for an efficient utilization of the message payload as most signals are up to a few bytes in length.

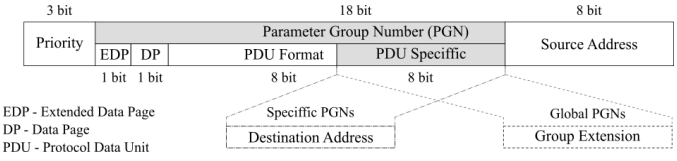

Fig. 2. J1939 breakdown of the extended CAN identifier field

The PGN field is composed of two bits denoting the data page, a Protocol Data Unit (PDU) format and a PDU specific field. Currently only two data pages are used in J1939 selectable by the Data Page bit while the Extended Data Page bit should always be set to 0. The PDU format field determines the format of the PDU and is one of the fields used to determine the PGN. If the value of the PDU field is below 240 (PDU1 format) then the PDU specific field is interpreted as a destination address. If its value is 240 to 255 (PDU2 format), then it is considered to be a group extension. The destination address defines the specific recipient of the message. Note that, even though by the nature of the CAN protocol all nodes will receive the message, it should be ignored by all except for the designated recipient. To send a message to all nodes the global destination address 255 should be used in this field. Sending destination specific messages is not possible when using PDU2 format.

J1939-71 defines signals and PGNs for commonly used functionalities [23] along with their specific cycle times and default priority. Some PGNs are allocated for protocol specific operations such as requesting information from other nodes, acknowledging protocol services, address management and the transport protocol needed for multi-frame messages.

Request messages can be sent globally or to a specific destination. A special use-case for the request PGN functionality is the address allocation procedure specified by the J1939-81 Network management document [24]. Each device in a J1939 network is required to have a valid address and an unique name (denoted as NAME). Besides uniquely identifying each node in the network, the NAME also denotes information about the industry sector for which the device is meant, its functionality and manufacturer as shown in Figure 3. Before participating in network communication a node should successfully complete the address claim procedure. This is somewhat similar to the IPv4 address conflict detection mechanism [25]. This procedure is usually done at system start-up but it can also be done any time a node is added to an already operating network. A request message sent for the Address Claimed PGN should result in all destination nodes transmitting an Address Claimed message with the address they successfully claimed or with the null address (254) if they were not able to claim any address. In case a node has not yet attempted to claim an address it should do so at this point by sending an Address claimed message with the desired address as the source address and the NAME in the data field. Upon receiving this message, any node claiming the same address should send a similar message with their own NAME. The sender with the lowest value NAME wins the address claim and signals this by sending another claim message while the others have to signal their claim loss by sending a Cannot Claim Source Address message. Upon claiming failure, depending on their configuration, some nodes may attempt to claim another address while other nodes will be limited to trying to retry claiming the same address.

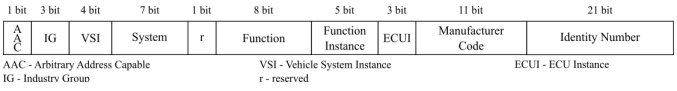

Fig. 3. Illustration of the NAME field

The SAE J1939 protocol allows the transmission of parameter groups of more than 8 bytes in length. Due to the nature of the CAN frame, the only way to accommodate this feature is to divide the data in several 8 byte chunks that can be sent individually in a CAN message. J1939 describes the Transport Protocol (TP) that should be used for this purpose. Two PGNs are used for this specific task: Transport Protocol Connection Management (TP.CM) and Transport Protocol Data Transfer (TP.DT). One of two transport protocol variants should be used depending on whether the multi packet message is sent to a specific or a global address:

1) Connection Mode Data Transfer - With this protocol variant the sender node initiates a connection to the receiver. The receiver can control the way in which the messages are sent. At any point the connection can be aborted by either the sender or receiver.
2) Broadcast Data Transfer - In this case messages are sent to all nodes and the message flow is only controlled by the sender.

When using the connection mode, several types of TP.CM messages are used:

- Request to Send (RTS) - sent by the transmitter to signal another node the intent to connect with it.
- Clear to Send (CTS) - flow control message used by the receiver in response to RTS and to specify the amount of data it can receive at a certain point.

- End of Message Acknowledgement (EoMsgAck) - used by the recipient of a large message to signal the transmitter that the entire message was successfully received.
- Connection Abort (ConnAbort) - can be used by either the receiver or the transmitter to signal the end of the connection.

The concept of Working Sets is introduced by the J1939 specification to serve distributed applications where several nodes act together reacting to messages sent to the same destination address to fulfil a single functionality. This feature was particularly employed for agricultural implements where several nodes need to cooperate as a single one. To set up the working set one node must assume the role of the master. The master node must first send a message that declares it as the master and specifies the number of members of the working set. This message is followed by a series of messages that nominate the other set members. After this set-up phase all working set members will listen to messages addressed to the master node. Additional logic is needed to distinguish between messages intended for the working set and messages specific to the master.

III. Security shortcomings of the J1939

All the specific vulnerabilities of the CAN protocol are also inherited by J1939. Probably the most important is a consequence of the arbitration mechanism which allows the possibility to mount a DoS attack by continuously sending frames with the identifier field set to the lowest possible value (0). By analyzing the J1939 protocol specification we were able to identify several shortcomings that leave room for interpretation and potential misuse. These specification flaws can be exploited to mount DoS and even masquerade attacks by only using the higher level J1939 protocol features. We continue by presenting the capabilities required to undertake malicious actions on the J1939 protocol and discuss each identified vulnerability in the following sections.

Some of the discussed vulnerabilities may not apply for systems that do not implement relevant parts of the J1939 protocol. For example, the J1939 support in the latest AUTOSAR (4.2.2) specification does not include the use of Request2 and Transfer PNGs [26] or the possibility to change node addresses (not even with a Commanded Address PNG) or NAMEs [27]. A more drastic example comes from the specification of the CAN interface for bodywork in Scania trucks [28] which states that large parts of the J1939 protocol are not implemented: no network management is used as addresses are statically allocated, requests, acknowledgements and commands are not supported and only one of the diagnostics messages is employed. Such practical deployments may be incidentally more secure, but here we focus on attacks that come from following standard specifications which should be the norm.

A. Attacker capabilities

We assume that the attacker node is able to gain access to the bus which connects the nodes targeted by the attack, either by compromising an existing node or by connecting as a new one. The malicious node should be capable of entirely altering its NAME and address according to the needs. It is also possible for the attacker to build any J1939-compliant message with complete control over all configurable message fields (eg. priority, PNG and address) and to transmit it even if it may be an out of order message (e.g., a doubled CTS when using connection mode for transmitting long messages). This node should be able to analyze all J1939 traffic without any message being filtered-out even when having other nodes as destination. These capabilities are withing reach as proved by numerous practical attacks and we demostrate the attacks by using an industry-standard CANoe simulation.

B. Disrupting address claims by genuine nodes

The standard specifies that the address claiming procedure should be completed by each node with a successful claim before allowing any other traffic from that node. Address claiming usually takes place in an initialization phase at power-on but it is allowed at any time in order to accommodate the case of subsequently added nodes. This will enable a malicious node to claim the address of an honest node and win it by declaring a NAME field with a lower value and hence a higher priority. By continuously preventing the honest node to obtain an address in the described manner, the malicious node would successfully mount a DoS attack. If directed to safety critical modules such as the engine control unit or the braking system while the vehicle is in operation, this attack could have serious consequences. The only safety measure mentioned by the specification is the use of address-to-NAME correspondence tables for safety critical modules to ensure they are assigned at some expected addresses. A malicious node could easily counteract this by declaring a valid NAME value which reflects the same field values as the node targeted for attack but with a lower identity number to ensure wining the address claim. Figure 4 illustrates the address claim attack on legit nodes that claim a single predefined address (a) and nodes that can dynamically select addresses (b). In both cases the legit node attempts to claim an address while the attacker prevents it by claiming the same address with a smaller value and therefore higher priority name, i.e. $Legit\ name > Fabricated\ name$.

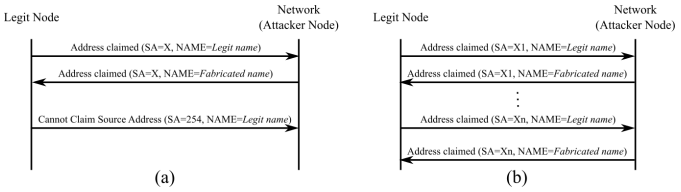

Fig. 4. Preventing address claiming: a) for a node which has a single address assigned to it, b) for a node that independently searches for an available address

Besides preventing a node to serve its function, by following this procedure, a malicious node could gain the identity of the targeted node and influence the vehicle operation by providing erroneous parameter values or sending control messages. Sending messages that should be sent by another node would also be possible without claiming its address with a valid NAME however, rendering the targeted node unable to communicate assures no interference from legit messages.

C. High busload by frequent requests for address claimed

When a global request for a PGNs is made, it should be answered by all the nodes that can provide an answer for it. The J1939 specification recommends sending at most 3 requests for a parameter group per second without making this mandatory or specifying countermeasures for requests sent more frequently. This allows a malicious node to mount Distributed DoS (DDoS) attacks by sending frequent requests that lead to an increased network traffic when responded to by all the recipients. The burst of responses will introduce delays for messages with lower priorities and higher value PGNs with a potential for being completely blocked if such requests are made frequently enough.

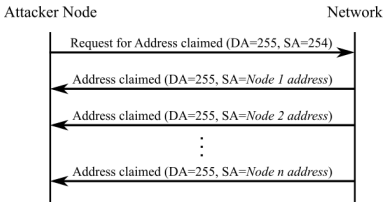

Fig. 5. The result of sending a Request for address claimed

We now consider the case of global requests for Address Claimed messages. Such a request has to be responded-to by all nodes on the bus with an Address claimed message generating an additional 8 byte data frame for each respondent as illustrated in Figure 5. The request as well as all the answers will have the destination address (DA) parameter set to the global address (255) and the source address (SA) set to either the address they have successfully claimed or the null address (254) if they did not yet claim an address. To ensure the request transmission at specific moments without being delayed by other higher priority traffic, the attacker could always send it with the highest priority (0). The effectiveness of such an attack greatly depends on the number of nodes and the communication speed and is based on the ability to generate a 100% bus load. For example, on a CAN bus with 30 nodes communicating at 250Kbit/s it will take at least 15.48ms (in the best case scenario when no bit stuffing is needed) for all the Address Claimed responses to be sent. In this case a DDoS attack can be mounted on traffic with lower priority than Address Claimed messages by sending a request for Address Claimed roughly every 15ms. If the same setup would have only 5 nodes then the attacker node would have to resend the request every 2.58ms. These two examples do not account for the existence of other traffic of higher priority that also contribute to the high bus load.

Similar attacks can be tailored for specific networks by analysing traffic and identifying requests that generate the most response traffic preferably with messages of highest possible priorities. Requests for multi-frame messages could also be considered but as the specified inter-packet interval can be up to 200ms the amount of resulting traffic would spread over a greater period in time reducing the influence on the bus load.

D. Transport protocol interruptions

The J1939 transport protocol allows the transmission of data packets longer than 8 bytes in size. While sending a message to a specific destination in connection mode, the connection can be aborted by both the sender and the receiver. An attacker node could use these features to interrupt the connection. To interrupt the connection a malicious node can send an abort message directed to either the sender or the receiver. If the abort message is sent to the receiver this node will cease to listen for further messages assuming the abort was issued by the legit sender, while the sender will finish the transmission of the previously specified number of frames and abort afterwards due to the lack of further control messages from the receiver. Alternatively, if the abort message is sent to the connection initiator, it will immediately stop sending further messages while the receiver will consider the connection interrupted due to timeout. Another way of generating a connection abort is for an attacker to send an additional clear to send control message after the one sent by the receiver.

E. Connection hijacking with impersonation by address claim

As a consequence of mounting an address claim attack on a node that is currently involved in a legit connection, the connection could be hijacked. To mount the attack, a malicious node must monitor the traffic and wait for the connection to be established, i.e. a RTS is transmitted by the initiator and responded to with a CTS. At this point the attacker should send an address claim message claiming the address of the node which it wishes to impersonate and declaring a higher priority NAME. Successfully claiming the address of the target node at this point assures that this node is prohibited from sending any messages in case it identifies transfer protocol misbehaviour. If it takes over the role of the initiator it will be able to send spoofed information to the receiver. In case it impersonates the receiver it could keep the connection blocked by continuously asking the initiator for data retransmission in a similar manner to the attack described in [6], the advantage in our approach being that it can prevent connections to a certain node even if a honest node has already managed to establish a connection with it.

F. Exclusion from working sets

A J1939 node cannot be a member of more than one working set at once. If a master defines a working set that includes a certain node which is already a member of another working set, then this node will end its membership in the previous working set and become a member of the newly defined one. A malicious node could use this behaviour to prevent a node from retaining membership of a legit working set and, as a consequence, block it from listening for messages sent to this working set.

IV. An authentication mechanism for J1939

All of the previous attacks have a unique cause: the lack of cryptographic authentication on genuine J1939 messages.

Consequently we now discuss a potential authentication protocol and later provide experimental evidence for its feasibility on a J1939 network. To remove all potential attacks, the protocol extends over all J1939 messages and not only over the restricted case of the previously discussed functionalities.

A. Protocol description

Given J1939 particularities, e.g., well-assigned functionalities for each node, as well as the nature of commercial vehicle manufacturing, where multiple producers contribute to a single vehicle, it is clear that authentication should rely on public key infrastructure (PKI). Only PKI can facilitate secure deployment of components in the vehicle without relying on secret keys that cannot be exchanged over the insecure CAN bus and are inconvenient to embed during manufacturing (as this will require the manufacturer to store secret keys for each vehicle incurring even higher security risks). Thus, each node must be in possession of a public-key certificate that is digitally signed by the OEM and by using this certificate it can also validate the public keys of the other nodes. The security designated ECU may be physically present as a distinct entity on the bus or its functionalities can be implemented by one of the existing nodes, e.g., the gateway node.

Briefly, we envision a 4 stage process as illustrated in Figure 6. First, in step 1, the OEM distributes signed certificates to the component manufacturer which are embedded and uniquely assigned to each ECU. These certificates are hard-coded on each ECU in step 2. Then in step 3 the OEM releases a security designated electronic control unit SeCU which has its own PKI certificate. Subsequently, in step 4 authenticated J1939 communication takes place inside the vehicle with initialization relying on the existing PKI infrastructure. We defer the discussion on specific concerns regarding the adoption of PKI on in-vehicle ECUs and related procedures for the following subsection, while here we stay to the protocol outline. For ECUs that can not handle public-key operations we consider that a token is used which links to a secret master key that will be used to exchange the ephemeral session keys. While using a symmetric master key can raise security concerns, for nodes that cannot handle PKI there is no alternative. The master-key will be hard-coded by the manufacturer on each ECU. The same key must be also known to SeCU a reason for which it is returned via a secure channel to the OEM in step 2' as a token to be hard-coded in the SeCU as well. Secure communication between the OEM and manufacturers is part of the usual production cycle so it is out of scope to discuss how this key will securely reach the OEM. We consider the secret-key based version of the protocol only as a transition step until all ECUs inside a car will be capable of public-key operations.

The protocol that we envision, J1939–ACAN, relies on two components: a more demanding initialization sub-protocol Init–J1939–ACAN and a faster, real-time authentication sub-protocol RunTime–J1939–ACAN. While Init–J1939–ACAN is demanding from a computational perspective we assume that this procedure is done rarely, i.e., when the car exits production, when a component is replaced or if for some

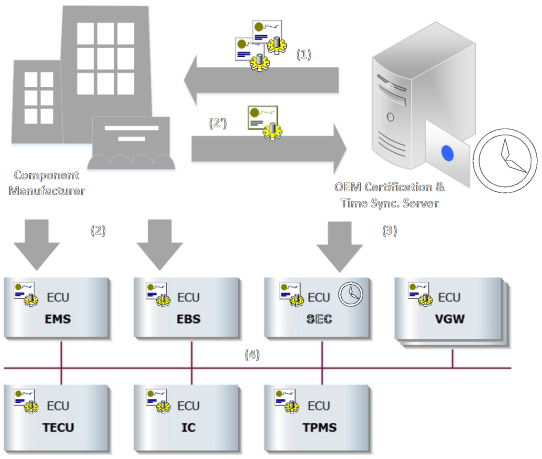

Fig. 6. Security setup and stages: (1) the OEM releases security designated ECU, (2) release certificates for component manufacturer (signed by OEM), (3) ECUs with signed certificates are deployed in the vehicle and (4) J1939 authentication communication takes place

TABLE I
SUMMARY OF NOTATIONS

$\mathcal{KD}$	key derivation process
Sig	digital signature
e	public key encryption
MAC	authentication tag, e.g., computed via HMAC
t_{SeCU}	current time on the security designated ECU
t_{snd}	time on the sender ECU
t_{rcv}	time on the receiver ECU
cnt_i	individual counter for each sender ECU
rnd	random material
$\twoheadrightarrow$	broadcast, i.e., message sent to all nodes
$\rightarrow$	unicast, i.e., message sent to individual node
$\mathsf{Cert}_{\mathsf{OEM}}$	certificate of the OEM
$\mathsf{Cert}_{\mathsf{ECU}_i}$	certificate of $\mathsf{ECU}_i, i = 1..n$
ECU_i	the i-th ECU with $i = 1..n$
SeCU	security designated ECU
K_{ses}	the secret session key
K_{sync}	a secret key for time-syncronization
K_m	the secret master-key

security reasons a new session key is desired. In case one node loses synchronization, the synchronization steps of Init–J1939–ACAN can be individually run between the node and the security designated controller SeCU. The synchronization steps are simple and rely only on symmetric cryptographic functions that require very little computational and communication overhead. Under normal running conditions, only the RunTime–J1939–ACAN is carried. In what follows we give the description of the two sub-protocols while in the experimental section we give experimental proofs on the feasibility of both components Init–J1939–ACAN and RunTime–J1939–ACAN. The protocol description that follows is generic and we discuss concrete instantiations for it in the following subsections.

We define the J1939–ACAN protocol as the following set of actions for each node:

I) Init–J1939–ACAN is the initialization stage which gathers procedures for broadcasting certificates, achieving time synchronization and establishing a session key. This procedure is run between each $\mathsf{ECU}_{i\in[1,n]}$ and the security designated controller SeCU in a sequential fashion as follows:

1) $\mathsf{SendCert}(\mathsf{ECU}_{i\in[1,n]}, \mathsf{SeCU})$ in which each ECU in pos-

session of a digital certificate announces his presence to the security designated controller SeCU by sending his certificate $\mathsf{Cert}_{\mathsf{ECU}_i}$ along with some fresh random value rnd'_i to assure freshness:

$$\mathsf{ECU}_{i\in[1,n]} \rightarrow \mathsf{SeCU} : \mathsf{Cert}_{\mathsf{ECU}_i}, rnd'_i$$

2) $\mathsf{SendSKey}(\mathsf{SeCU}, \mathsf{ECU}_{i\in[1,n]})$ in which the security designated controller SeCU sends to each $\mathsf{ECU}, i \in [1,n]$ the current session key and a synchronization key encrypted with the the ECU's public key. These keys will be sent only to ECUs that have proved that they are authorized for this network by previously showing their signed certificate:

$$\mathsf{SeCU} \rightarrow \mathsf{ECU}_{i\in[1,n]} : e_{pbk_{\mathsf{ECU}_i}}(\mathrm{K}_{ses}, \mathrm{K}_{sync}), \\ Sig_{\mathsf{SeCU}}(e_{pbk_{\mathsf{ECU}_i}}(\mathrm{K}_{ses}, \mathrm{K}_{sync}), rnd'_i)$$

where the session key K_{ses} is a randomly generated session key common for all nodes that are authorized for the network and K_{sync} is a random key that is uniquely generated for each ECU and used for time synchronization. The content is signed to ensure the ECUs about the origin of the packet and includes rnd'_i to prove the freshness of the keys.

3) $\mathsf{SendToken}(\mathsf{ECU}_{i\in[1,n]}, \mathsf{SeCU})$ in which each ECU (that does not own a certificate) announces his presence to the security designated ECU by sending the token along with some fresh random value rnd_i to assure freshness of the forthcoming session key:

$$\mathsf{ECU}_{i\in[1,n]} \rightarrow \mathsf{SeCU} : \mathrm{Token}, rnd'_i$$

We assume that the token contains an identifier for the node, an identifier for the network that he belongs to and information on the master key. All these can be easily encrypted by the OEM in the token with a key that is known to SeCU.

4) $\mathsf{SendSKeySym}(\mathsf{SeCU}, \mathsf{ECU}_{i\in[1,n]})$ in which the security designated controller SeCU sends to each $\mathsf{ECU}, i \in [1,n]$ (that does not own a public certificate and has instead presented a token) the current session key and a synchronization key encrypted with the the ECU's symmetric master key:

$$\mathsf{SeCU} \rightarrow \mathsf{ECU}_{i\in[1,n]} : e_{\mathrm{K}_m}(\mathrm{K}_{ses}, \mathrm{K}_{sync}, rnd'_i)$$

These keys will be sent only to ECUs that had their tokens registred to the OEM and were authorized to be part of the network.

5) $\mathsf{RqTimeSync}(\mathsf{ECU}_{i\in 1..n}, \mathsf{SeCU})$ is the procedure in which each ECU requests a time synchronization to the security designated controller SeCU by sending a fresh random value along with a MAC computed on it with the synchronization key:

$$\mathsf{ECU}_{i\in[1,n]} \twoheadrightarrow \mathsf{SeCU} : rnd''_i, MAC_{\mathrm{K}_{sync}}(rnd''_i)$$

6) $\mathsf{SendTime}(\mathsf{SeCU}, \mathsf{ECU}_{i\in 1..n})$ in which the security designated controller SeCU sends the current time and authenticates it with a MAC computed with the synchronization key along with the random value received from the ECU:

$$\mathsf{SeCU} \twoheadrightarrow \mathsf{ECU}_{i\in[1,n]} : t_{\mathsf{SeCU}}, MAC_{\mathrm{K}_{sync}}(t_{\mathsf{SeCU}}, rnd''_i)$$

for correct establishment of the current time, each $\mathsf{ECU}, i \in [1,n]$ sets the current time as $t_{\mathsf{SeCU}} + \epsilon_i$ when receiving t_{SeCU} where ϵ_i is the synchronization error of ECU_i computed as the time elapsed from sending rnd''_i in step $\mathsf{SendCert}(\mathsf{ECU}_i)$ up until receiving t_{SeCU} in step $\mathsf{SendTime}(\mathsf{SeCU})$ (note that at this point the current time at SeCU is in the interval $[t_{\mathsf{SeCU}}, t_{\mathsf{SeCU}} + \epsilon_i]$).

II) RunTime–J1939–ACAN is the runtime stage in which the secret session key is used to provide authentication on all messages including the address claiming procedure SndAddrClaim and requesting for an abort of the transport protocol TP.Conn_Abort:

1) $\mathsf{SendJ1939}(\mathsf{ECU}_i, \mathsf{ECU}_j \vee \mathsf{All})$ is the regular sending procedure for J1939 messages, denoted by $\mathrm{m}_{\mathrm{j1939}}$, no modifications required.
2) $\mathsf{SendTagJ1939}(\mathsf{ECU}_i, \mathsf{ECU}_j \vee \mathsf{All})$ by which ECU_i sends an accompanying authentication tag in addition to any of the regular J1939 messages $\mathrm{m}_{\mathrm{j1939}}$ along with the current time-stamp:

$$\mathsf{ECU}_{i=1..n} \twoheadrightarrow \mathsf{All} : ID_m, t_{snd}, \\ cnt_i, MAC_{\mathrm{K}_{ses}}(\mathrm{m}_{\mathrm{j1939}}, t_{snd}, cnt_i)$$

here by $\mathrm{m}_{\mathrm{j1939}}$ we denote the entire J1939 message for which the tag is being constructed (i.e., the identifier and data fields of this message), and by ID_m the identifier of the message being authenticated.

As a general rule over protocol messages, any $\mathsf{ECU}_{i\in[1,n]}$ ignores any requests from $\mathsf{ECU}_{j\in[1,n]}$, e.g., $\mathsf{TP.Conn\_Abort}(\mathsf{ECU}_j)$ or $\mathsf{SndAddrClaim}(\mathsf{ECU}_j)$, in case when the request has failed the authentication test. Checking for authenticity implies, besides verification of the authentication tag, that each node checks for freshness of the message by validating that $|t_{snd} - t_{rcv}| < \delta + \epsilon$, where δ is a fixed propagation delay and ϵ the synchronization error, and that $cnt_i > cnt'_i$ where cnt'_i is the last received message counter from ECU_i.

A short motivation on the use of both sender time t_{snd} and local counter cnt_i for each authentication tag is in order. To check for freshness the receiver will need to verify that $|t_{snd} - t_{rcv}| < \delta + \epsilon$, as he needs to account for both the propagation delay δ and synchronization errors. This opens door for a reply attack as a message along with its tag $MAC_{\mathrm{K}_{ses}}(\mathrm{m}_{\mathrm{j1939}}, t_{snd})$ can be replayed for as long as $|t_{snd} - t_{rcv}| < \delta + \epsilon$ holds (note omission of the counter in this example). To avoid this, while still allowing burst periods from honest ECUs we use an individual counter cnt_i that is incremented each time a new frame is sent. For each of the received messages, the receiver must check that $cnt_i > cnt'_i$ where cnt'_i is the last

received message counter from ECU_i. This way it is possible to discern between burst periods in which frames are sent at short time intervals (shorter than the propagation delay plus the synchronization error) and replay attacks. The counter can be allowed to reset when the maximum value is reached, we simply need to account for messages that are sent during a very short period of $\delta + \epsilon$ (i.e., an interval in which the counter simply must not repeat). We use 32 bits for this counter in our experiments with CAN-FD, but much shorter values can be used (a 10 bit counter is proposed for the 2 frame CAN authentication). By experiments (as detailed in a forthcoming section) we determined that the shortest delays between two frames, i.e., in a burst period, is around a dozen microseconds. The use of time-stamps alone would not be sufficient as the synchronization errors are usually higher than this.

B. Feasibility of PKI and related procedures

The adoption of PKI is unavoidable for assuring fresh session keys between components that do not share a common production history. This is generally the case in the absence of a secure environment were the same secret shared keys can be imprinted to all components. While it is clear that not all ECUs inside contemporary cars are able to handle public-key cryptography, we provide arguments on why the adoption of PKI should be a realistic step for in-vehicle ECUs. There are three main arguments as we point out next: the requirement for PKI in inter-vehicular networks, the support for public-key operations in the AUTOSAR cryptographic specifications [29], [30] and the spread of open-source libraries for embedded devices that provide extensive support for public-key cryptography.

Relying on PKI is not an unusual demand for the automotive industry and is clearly within reach since numerous modern applications related to vehicle-to-vehicle or vehicle-to-infrastructure communication are depending on this as well. There a number of research works that address this [31], [32], [33], [34] and recently the Office of the Federal Register has published a request for information on credential management systems for vehicle to vehicle communication [35]. All these specifically address procedures for certificate distribution and certificate revocation, etc. Thus the related infrastructure will be in place at least for V2X communication modules. This makes the main requirement of our protocol for a security designated unit SeCU that supports PKI to be realistic.

For all other ECUs except SeCU, provided that they have sufficient computational power, the support for public-key operations is immediate from both existing standards or existing cryptographic libraries for embedded devices. Indeed, current specifications for automotive-grade cryptographic libraries inside the AUTOSAR standard [29], [30] provide interfaces for all asymmetric functionalities: encryption, signing and key exchange. Thus, mere compliance with current standards will assure support for public-key operations. Also, open source cryptographic libraries provide lots of support for such functionalities. For example, the WolfSSL library that we use in the experimental section [36] provides support for DSA, Diffie-Hellman key-exchange and their elliptical-curve derivatives.

As for PKI related functionalities such as certificate issuing and revocation these will go in the usual way. For example, certificate revocation will be based on certificate revocation lists (CRL). The security designated ECU can maintain an updated list of revoked certificates by remote connectivity to the OEM server. Remote connectivity via 3G is already common inside cars. The size of the list should not cause concerns since SeCU can store only the IDs of the certificate, or their hash value. Once the certificate of a node is revoked it must be immediately changed. Since this requires an update to the software of the corresponding ECU, given the nature of the automotive domain this should be done in an authorized garage. A remote procedure may be also put in place via secure channels such as SSL/TLS. But it is out of scope for the current work to illustrate specific procedures. Nonetheless, PKI related procedures are well studied in the literature and many current proposals exist, e.g., [34], which can be modified to specific needs or even straight-forwardly adopted.

C. Practical instantiations of Init–J1939–ACAN

Using standard CAN frames to send messages specific to the proposed authentication protocol would require some trade-offs between performance and security as the 8 byte payload of a single CAN frame is not sufficient to send large messages containing signatures or bigger authentication tags. Since there exists active interest in mapping J1939 messages to CAN-FD frames [37], we find it appropriate to study the possible use of CAN-FD for providing J1939 authentication while also discussing solutions for coping with the limitations of CAN frames.

By proper choice of parameters, the 64 byte frame of CAN-FD has the appropriate size for carrying signed messages. An RSA signature leads to 1024-2048 bits, this is too much even for CAN-FD frames. A DSA signature however, by proper choice of parameters, could fit in one CAN-FD frame. According to current specifications on key sizes [38], the DSA secret key size is set at 224 bits over a 2048 bit modulus for a validity from 2011-2030. Since the size of the signature is double this size, it leads to 448 bits which leaves room for an extra 64 bits. That is, one CAN-FD frame could carry one signature plus the existing 64 bit CAN data. This ensures a good implementation perspective. As for beyond 2030, a key of 256 bits is needed for DSA and a modulus of 3072 bits instead of 2048 bits. Alternatively, a 256 bits elliptical curve will attain the same security at a more compact size reducing frame and memory overheads. These are manageable by high-end cores as we argue in the experimental section were we provide experimental results.

We now provide concrete quantifications for the size of the packets involved in the handshake to set way for performance evaluation of Init–J1939–ACAN. The following quantifications are based around standard DSA signatures which can be further improved by ECC. The hand-shake implied by Init–J1939–ACAN is suggested in Figure 7 for nodes supporting public-key operations and in Figure 8 for non-PKI nodes. On each arrow corresponding to a sent message the required CAN-FD or CAN packets are numbered. It feels natural to assume

that nodes supporting PKI will be present on the newer CAN-FD layer. The image can be immediately adapted for the case of regular CAN but the number of packets increases by the expected factor of 8. Similarly for the case of non-PKI nodes we assume a regular CAN bus but it straight-forward to port this on the larger CAN-FD frames and the overhead will be reduced to 1 frame for each round of the protocol.

The use of DSA allows us to take advantage of fixed parameters for the entire system. In case of operations over Z_p the common parameters are the modulus p, the generator g and q which is the divider of $p-1$. The same holds for the use of elliptical curves since all nodes can use the same curve. Thus we consider that all nodes share these parameters and they don't need to be exchanged over the network. Once our setup is fixed around DSA signatures, the usual way to obtain an asymmetric encryption (required by the SendSKey step) is to rely on the Diffie-Hellman key-exchange [39] protocol. To avoid man-in-the-middle attacks, which are feasible on the basic Diffie-Hellman key-exchange, we prefer to modify one of its immediate derivatives, the security enhanced STS protocol [40]. The modifications consist in reducing the certificate size, which in practical instantiations could reach several kilobytes, to a certificate that contains the public key, some information from the OEM that confirm that this is the intended network for the ECU, e.g., a network and node ID, and the signature of the security controller SeCU (this signature will be embedded by the OEM which is the issuer of SeCU). In principle, this is a pre-shared public-key version of the Diffie-Hellman key-exchange similar to the ElGamal encryption [41] and helps both in avoiding man-in-the-middle attacks and in reducing the size of the certificate. As for symmetric encryption in what follows we consider that AES with 128 bit key is used, while for the MAC the HMAC construction with SHA256. Experimental results will be provided for all these primitives.

Frame allocation with DSA and ECC versions. We now explain frame allocation as presented in Figure 7. In the first step each node will have to announce his public-key certificate, i.e., the Diffie-Hellman key share $g^x \mathrm{mod} p$ along with the information that proves it is part of the network and the signature of the OEM (the security designated controller SeCU is able to verify such signatures). For a 2048 bit modulus the value of $g^x \mathrm{mod} p$ will require 4 CAN-FD frames, i.e., frames f_1–f_4. The additional information is present along with some random material rnd' is included in frame f_5. For the SHA256 based DSA signature, two 256 bits values are generated during signing and these fit in the 512 bit frame, thus frame f_6 holds this signature of the OEM. In the second step the security designated controller SeCU replies with the freshly generated $g^y \mathrm{mod} p$ which again takes 4 frames, i.e., frames f_7–f_{10}. The Diffie-Hellman session key is now $K = g^{xy} \mathrm{mod} p$ and this is used to encrypt a signature and the two keys K_{ses}, K_{sync}. Thus, frame f_{11} is needed to carry the DSA signature and frame f_{12} to carry the encrypted keys K_{ses}, K_{sync}. Encrypting this payload causes only small computational overheads and requires no additional bandwidth. For security reasons a CBC mode of operation on AES can be used which requires 128 bits for the initialization vector IV and fortunately there is enough room for this since the two secret keys K_{ses}, K_{sync} require only 256 bits out of the 512 bits of the frame. Then the ECU responds with a fresh random value rnd'' plus a MAC computed with the synchronization key K_{sync} which require one more frame, i.e., f_{13}. Finally, the security controller answers with the current time and a MAC which fit in frame f_{14}. We consider a time kept on 64 bits and a MAC of 256 bits or less which is secure enough for our purpose. Since a compact 256-bit curve offers more or similar security to the 2048-bit modulus, the size of the shares g^x and g^y decreases to the coordinates of these points on the curve making a reduction from 4 frames to 2. Instead of 14 frames, 10 frames will be sufficient for the ECC based initialization. Consequently, the public-key version of Init–J1939–ACAN requires 10–14 CAN-FD frames.

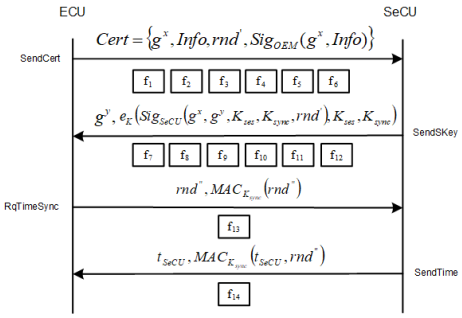

Fig. 7. Authentication-synchronization sequence (STS-based protocol) for nodes supporting public-key operations (CAN-FD frames)

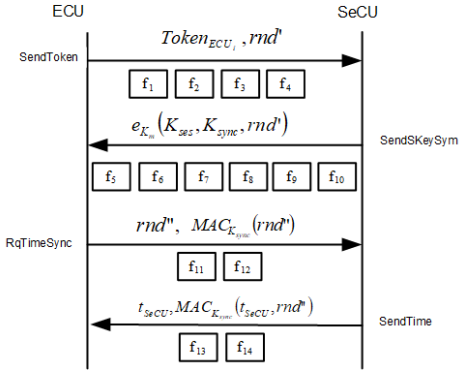

Fig. 8. Authentication sequence for non-PKI nodes (regular CAN frames)

Frame allocation for non-PKI nodes. We explain frame allocation for non-PKI compliant nodes as presented in Figure 8. For nodes not supporting public-key operations, the first message consists in the token along with a first random value rnd'. We consider the token to be 192 bits and the random value 64 bits, requiring a total of 4 frames, i.e., f_1–f_4. The 192-bit token can hold an identifier for the node, an identifier for the associated network and a link to the master key or the master keys itself - all these encrypted with a key known only to the OEM and SeCU. For all these 192 bits seem sufficient. Subsequently the security controller SeCU replies with the encrypted session key and the synchronization key. Since encryption must be performed in a secure mode of operation such as CBC, the first frame, i.e., frame f_5, must

hold an initialization value or counter for which 64 bits are sufficient. Assuming the keys K_{ses} and K_{sync} are 128 bits each they require 4 more frames, i.e., f_6–f_9. To ensure freshness for the response the random value from the previous round must be also included, hence the requirement for frame f_{10}. Subsequently, a new random value is sent for time synchronization along with a MAC. Considering that MACs can be safely truncated to 64 bits we need two more frames for the random value and the MAC, i.e., f_{11}–f_{12}. Finally, the current time represented on 64 bits along with a new MAC on it, again truncated to 64, require two more frames, i.e., f_{13}–f_{14}.

D. Practical instation of RunTime–J1939–ACAN

Since the initialization stage of the protocol Init − J1939–ACAN is done only once at start-up, despite depending on a high number of CAN frames, it will not impede regular traffic. Of more concern for real-time needs is the authentication during the run-time stage RunTime–J1939–ACAN.

We propose the following practical instantiation when using CAN-FD: 24 bytes for the authentication data field containing 29 bits for the ID of the authenticated message, 64 bits for the authentication tag, 64 bits for the time-stamp, 32 bits for the counter and 3 additional padding bits to obtain the 8 bit alignment.

Figure 9 (a) illustrates the authentication message components as organized in the data field of a CAN-FD frame and suggests their distribution into three CAN frames. Sending the authentication information through CAN frames will lead to a considerable increase in busload since for every J1939 message 3 additional CAN frames would be needed. An improvement could be made, at the cost of a reduced security level, by decreasing the size of the authentication message components so that it can be sent through only 2 CAN frames as illustrated in Figure 9 (b).

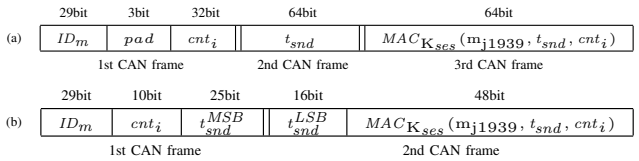

Fig. 9. Full authentication message organization in a CAN-FD frame and indication on data splitting between three CAN frames (a). Reduced size authentication message split in two CAN frames (b)

E. Message allocation according to J1939 specifications

Introducing security to the J1939 protocol would require sending security-related information, such as authentication or key exchange data, using messages that comply with the protocol specification. Since security-related messages were not included in the J1939 protocol (besides the ones used for diagnostic functions), specific messages will have to be defined for this purpose. This will call for the allocation of dedicated PGNs to identify the information transmitted within these messages. Both global and specific addressing would be needed depending on the nature of the messages to which the security mechanism is applied. For example an authentication data generated for a specific address message would also have to be sent with a specific address. The same applies to a global message for which an authentication message should also be global. This limits the choice of PGNs since some can only be used for global addressing. One possible approach would be to use the proprietary PGNs defined by the J1939 specification but there are only two such PGNs (Proprietary A and Proprietary A1) that can be used for specific addressing which might not be enough to support for all needed security services. On the other hand, the proprietary PGNs were assigned for producer specific functionalities while the J1939 specification recommends requests for adding PGNs of general interest to the list of predefined PGNs. This suggests a second, more suitable approach which is to reserve new PGNs in the specification for security services as they would be of general interest for all manufacturers.

An interely different approach suitable for sending the authentication tag, which is applicable when using CAN-FD frames, would be to send the authentication data along with the corresponding message in a single frame. This would be possible since the CAN-FD data field can accommodate up to 64 bytes. However, using this approach would offer no backwards compatibility with current implementations of the J1939 protocol.

V. Experimental results

The feasibility of the presented attacks and the effect of introducing the proposed mechanism were evaluated using the simulation environment available in CANoe, a comprehensive tool for the analysis, development and testing of automotive networks.

A. Setup

We built our tests on the J1939 system demo setup provided in the CANoe environment [42]. The setup consists in a simulated sub-network bus with 6 simulated nodes each implementing basic vehicle functionalities: Engine Management System (EMS), Electronic Brake system (EBS), Vehicle GateWay (VGW), Transmission Electronic Control Unit (TECU), Instrument Cluster (IC) and Tire Pressure Monitoring System (TPMS) as depicted by Figure 10. The VGW node is responssible for relaying the traffic to and from other vehicle subnetworks. This subnetwork setup is consistent with real-life architectures such as the networks employed by Scania trucks [43] in which subnetworks consist of 3 to 7 nodes. The communication bit rate is set at 250Kbit/s, while the network nodes and associated messages with corresponding PNGs are defined as recommended by the J1939 specification [23]. The specific behaviour of each node is implemented in CANoe using CAPL scripts while the J1939-related behaviour is handled by the J1939 interaction layer which assures protocol compliant message mapping and transmission behaviour (eg. address claiming and transfer protocol).

We implemented the behaviour needed to evaluate the J1939 protocol vulnerabilities in CAPL scripts using support from the J1939 interaction layer and added new messages where needed. The resulting network behaviour was analysed by studying the CANoe message trace after running the simulation.

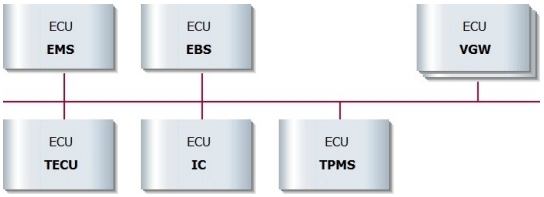

Fig. 10. CANoe simulation powertrain J1939 network

B. Testing J1939 vulnerabilities by industry-standard simulations

Address claim DoS. During the simulation startup each node tries to claim its associated address. Our address claim DoS attack implementation claims the addresses already claimed by other nodes by declaring lower value addresses. Running this attack on the simulation succeeded in overtaking addresses of other nodes and preventing them to successfully claim an address.

Request for address claim DDoS. For mounting the DDoS attack using the request for address claim approach we used a node that cyclically sent the request. To evaluate the attack effects we monitored the bus load depending on the request cycle time. The normal simulation bus load stays below 20% without any traffic generated by the attack. Sending the request for address claimed message cyclically at 100ms the bus load rises to around 22%. At this point the CANoe trace already informs that the request is sent more often than recommended by the J1939 spec (more than 3 times/s). Changing the cycle time to 10ms increases the busload to 52% while at 4ms the busload increases to 88% causing more than half of the messages not to be sent due to disruptions in prescribed cycle times.

Transport protocol DoS. For testing the attacks on the TP we were not able to use the CANoe J1939 layer functionality as it automatically handles transmission of messages longer than 8 bytes. Our attempts to send TP-related messages using J1939-specific functions resulted in errors being displayed and no actual message being sent. We therefore had to mount our attack using the CAN layer. Our attacker routine was designed to wait for the Clear to Send signal from the receiver of a specific message and send an abort message (declaring the reason as insufficient resources) immediately after it.

When the abort message is directed to the message originator the sender manages to send the first data frame in the intended sequence before interpreting the abort command. The receiver does not receive all the message parts within the expected time and sends another abort message due to timeout. In this case the CANoe trace identifies the sent data frame and the second abort frame as violations to the J1939 protocol. If the abort message has the receiver as the destination, the sender sends all message parts specified in the CTS message (6 in our case) while the receiver immediately declares the connection as closed. For this test CANoe identified all the data frames as protocol violations.

We also tried using an additional CTS to cause the connection abortion and found that if the injected CTS was the first to reach the bus, the legit receiver waited for the transmitter to finish serving this CTS before sending its own CTS message which will not be seen as invalid, even though it asks for the same data, as it is considered as a retry request. When the injected CTS is sent after the legit one the transmitter manages to send one message in the expected sequence after which it immediately sends an abort message with reason code 0xFF (undefined reason). The injected CTS frame is in this case seen as an invalid occurrence in respect to the J1939 protocol. While trying out various CTS parameters we came upon an undefined behaviour in the case when CTS specifies that 0 messages should be sent. Upon sending this message the sender will send one sequence from the large message starting from the one specified by the CTS but this will be reported as a J1939 protocol violation in the CANoe trace.

Connection hijacking. We tested the connection hijacking by having a node monitor bus traffic for transfer protocol connections and making an address claim for the address of either one of the connected parties. The result, in both cases, was that the target node transmitted a message indicating that it cannot claim an address followed by a connection abort message with reason 0xFF (undefined). Aborting the connection upon loosing the claim on the address is the logical approach, however, this behaviour is in contradiction with the specification since it is clearly stated that a J1939 node cannot transmit messages other then address claims unless they are successful in claiming an address.

Exclusion from working sets. We found that the CANoe J1939 interaction layer has no integrated support for automatic handling of working set messages. Therefore, we could not test this attack using the simulation.

Attacks on authenticated traffic. When employing the proposed authentication scheme nodes will process a received message only if authentication succeeds. Otherwise they will ignore and discard the message. Therefore, any of the attempts to mount the described attacks fail since the attacker node is not a legit network node and has no ability to send authenticated messages.

DoS attacks. We do note that all of the previously discussed attacks are a form of DoS. The security mechanisms that we provided are concerned with protection against attacks caused by impersonations, exclusion from working sets, disrupting address claims, etc., which may be viewed as DoS attacks at a logical level. However we are not concerned with resource-exhaustion DoS attacks caused by a more or less powerful node. DoS by resource-exhaustion cannot be prevented since an adversarial node can always write dominant bits on the bus preventing the other nodes from gaining access to the bus (as a resource). Moreover it can induce unnecessary cryptographic operations, e.g., signature or MAC verifications that fail by altering frames or injecting frames of his own. Such attacks cannot be prevented by cryptographic security.

C. Computational overheads of cryptographic primitives

We considered for evaluation a set of 5 platforms from the low (RL-78, S12), mid (MPC5606) and high-performance (TC297, RH850) areas. The scope was to determine how each category copes with the computational requirements of the proposed scheme.

TABLE II
COMPUTATIONAL OVERHEADS OF PUBLIC-KEY OPERATIONS ON TC297

Memory allocation	Diffie-Hellman		DSA2048		ECDSA256	
	Key gen	Key agree	Sign	Verif	Sign	Verif
Static	$540ms$	$550ms$	$240ms$	$490ms$	N/A	N/A
Dynamic	$1.02s$	$1.06s$	$300ms$	$580ms$	$88ms$	$122ms$

TABLE III
COMPUTATIONAL OVERHEADS FOR SYMMETRIC PRIMITIVES

Cryptographic primitive	Platform				
	RL78 D1A	S12XDT	MPC5606B	TC297	RH850G3M
SHA1	$6.15ms$	$2.94ms$	$578.75\mu s$	$14\mu s$	$56.95\mu s$
SHA256	$2.20ms$	$6.21ms$	$563.75\mu s$	$41\mu s$	$37.54\mu s$
SHA3-256	$120.9ms$	$46.55ms$	$19.13ms$	$66\mu s$	$2.57ms$
AES-128	$0.71ms$	$0.40ms$	$427\mu s$	$18\mu s$	$42.03\mu s$

For the asymmetric cryptographic primitives required by the Init−J1939–ACAN stage we used wolfSSL (version 3.12.0) [36], an embedded SSL library which provides a wide range of cryptographic algorithms. From this library we used the Diffie-Hellman, DSA and ECDSA implementations. Unsurprisingly, while trying to port these on our target platforms, we found that only the high-end devices could cope with public key cryptography. The other platforms failed in providing enough RAM memory (and Flash in some cases) to accommodate the implementations. High-end microcontrollers were ready for running asymmetric primitives while the low and mid-end side could achieve this only if equipped with dedicated hardware modules. Larger memories are available on low-end cores but computational performance will still be considerably lower.

As a reference for the high-end devices performance, we provide experimental results obtained on the AURIX TC297 microcontroller in Table II. Here, the Diffie-Hellman *Key gen* stands for the session key generation while *Key agree* stands for the shared key derivation (computationally speaking these 2 operations are equivalent). We evaluated both static and dynamic memory allocation settings of the library to illustrate effects of the speed-memory tradeoff.

For the devices not capable of handling public key primitives we evaluated the overhead of using AES-128. The last line in Table III shows the time needed to encrypt one AES-128 block on each platform.

We give the computational overheads for generating hashes over J1939 messages on the evaluated automotive platforms in Table III. The experimental data for each platform was obtained using a 512 bit input (which fits for the size of the extended ID, 8 byte data field, time-stamp, counter and key).

Execution times for HMACs can be easily inferred based on this table since HMAC involves exactly two hash operations, i.e., $HMAC(K,m) = H((K^{'} \oplus opad)||H((K^{'} \oplus ipad)||m))$. On the low-end RL78 and S12 platforms, authentication delays given by the computational needs are in the order of milliseconds less than 10 ms for SHA1 or SHA2 and less than 150 ms for SHA3. This would make it possible to sustain authenticated communication only for periodic messages that have cycles larger than this computational effort. In the case of the other platforms sending authenticated messages via SHA1 or SHA2 based HMAC will not be problematic. The newer standard SHA3 seems to be quite demanding and likely not suitable for the real-time nature of in-vehicle communication. Fortunately SHA256 offers sufficient security when used in HMAC.

TABLE IV
INIT-J1939-ACAN PER NODE COMMUNICATION OVERHEAD

Bit rate	CAN	CAN-FD	
		Data rate: 1Mbit/s	Data rate: 4Mbit/s
250Kbit/s	$7.28ms$	$10.86ms$	$4.02ms$
500Kbit/s	$3.64ms$	$10.00ms$	$3.15ms$

D. Performance evaluation of Init–J1939–ACAN

The initialization stage consists in the transmission of four messages for each node on the bus (besides the security ECU): SendCert/SendToken, SendSKey/SendSKeySym, RqTimeSync and SendTime. Table IV holds the maximum communication overheads (considering the maximum number of stuffing bits) involved in running the initialization stage for one node depending on the employed protocol. This requires the transmission of 14 CAN frames (for non-PKI nodes as per Figure 8) or 14 CAN-FD frames (for PKI nodes as per Figure 7) with a data field of 8 and 64 bytes respectively. For CAN-FD each row indicates the arbitration phase bit rate while the separate columns include results for 1 and 4Mbit/s data phase bit rates.

In Figure 11 we depict overheads for a regular CAN bus, while in Figure 12 for a CAN-FD bus with 250kbps bit rate during arbitration. The overheads are depicted based on the computational time and bus delays that were previously determined based on experimental measurements. As a general rule for all these figures, the plots on the left are showing the time required to send the initialization frames on the bus alone, while the plots on the right depict the complete initialization time which also accounts for the computational time. The computational overheads induce more time than the busload, especially in the case of the more expensive public-key operations. The number of computations that we account for follows directly from the proposed instantiations of the protocol in Figures 7 and 8. In the symmetric setup, these sum up to 2 MAC computations and 2 MAC verifications and 2 decrypted blocks for frames f_1–f_3 then 3 encrypted/decrypted blocks for frames f_5–f_{10}. In the asymmetric setup, the higher costs are induced by the public key-encryption (step 2), the 2 signature verifications (steps 1 and 2) and signing (step 2). Two MAC computations and verifications are also added to this. Moreover encryption extends over frames f_{11}–f_{12} and since each frame carries 512 bits we have a total of 1024 bits which require 8 AES block encryptions/decryptions (this is still a very small amount of time compared to public-key operations).

CAN-based initialization considers a bus-load from 125kbps to 1Mbps and up to 32 nodes. Initialization time tops at around 2.1s in case of 32 nodes and 125kbps. Figures 11 (a) and 11 (b) show the transmission time and the full initialization

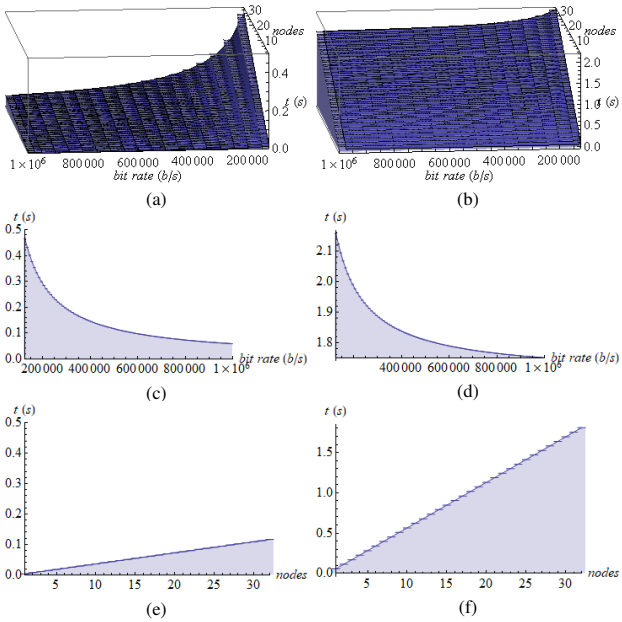

Fig. 11. Transmission time for non-PKI initialization frames on CAN: (a) variation of bus-time with bit rate and number of nodes, (b) variation of full initialization time with bit rate and number of nodes, (c) variation of bus-time with bit rate at 32 nodes, (d) variation of full initialization time with bit rate at 32 nodes, (e) variation of bus-time with the number of nodes at 500kbps bit rate and (f) variation of full initialization time with the number of nodes at 500kbps bit rate

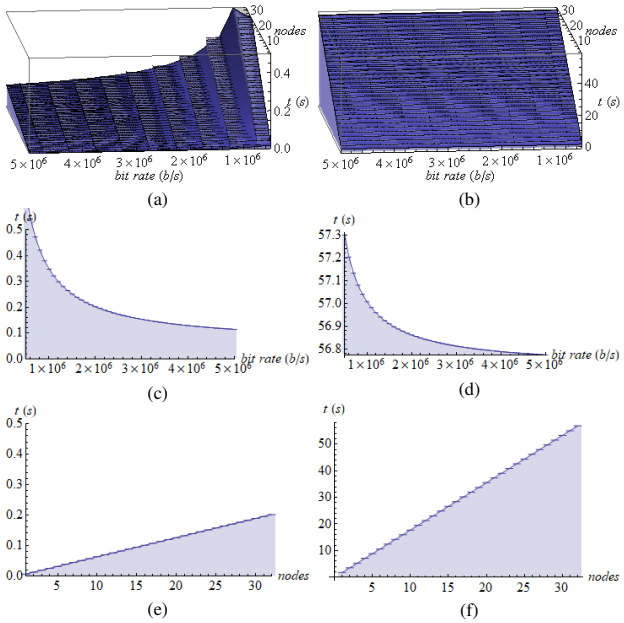

Fig. 12. Transmission time for PKI initialization frames on CAN-FD (250kps during arbitration): (a) variation of bus-time with bit rate and number of nodes (b), variation of full initialization time with bit rate and number of nodes, (c) variation of bus-time with bit rate at 32 nodes, (d) variation of full initialization time with bit rate at 32 nodes, (e) variation of bus-time with the number of nodes at 2Mbps bit rate and (f) variation of full initialization time with the number of nodes at 2Mbps bit rate

time (including computational overheads) as they vary with bandwidth and the number of nodes. Then in 11 (c) and (d) we keep the number of nodes constant at 32 and show the variation of the transmission time (c) and full initialization time (d) with the bandwidth. Finally in 11 (e) and (f) we keep the bandwidth constant at 500kbps and show the variation with the number of nodes. Since the number of computations increases linearly with the number of nodes it is expected that the highest delay of around 2.1s is achieved when the number of nodes reaches 32.

We structure similarly the depictions for CAN-FD in Figure 12. Again, 12 (a) and 12 (b) show the transmission time and the full initialization time (including computational overheads) as they vary with the data phase bit rate and the number of nodes. Then in 12 (c) and 12 (d) we keep the number of nodes constant at 32 while 12 (e) and 12 (f) we keep the data phase bit rate constant at 2Mbit/s. Due to the more intensive public-key operations, the initialization time tops at around 57.3s but this happens for a high number of nodes (32) and low data phase bit rate (1Mbps). This should be acceptable if initialization is done rarely, e.g., during production or in a specialized garage once a component is replaced.

We have also experimented with 500kbps during arbitration but the results were almost identical to the case of 250kbps. This is expected since it is the larger data-field that causes the overhead and it benefits from the extended bit-rate of CAN-FD regardless of the bit rate during arbitration. The initialization time is smaller for CAN than for CAN-FD, but CAN initialization is done without the more expensive public-key operation and thus the security implications are distinct.

E. Performance evaluation of RunTime–J1939–ACAN

In this section we discuss the impact of the run-time stage RunTime–J1939–ACAN of the authentication protocol on the bus-load and resulting delays. Distinct tests were made in which the authentication tag was sent within the same frame as the J1939 message and as a separate message over both CAN and CAN-FD frames. Since for the moment there is no support for J1939 over CAN-FD in CANoe, the CAN-FD authentication frames are sent using the CANoe CAN layer support in our experiments. The CAN-FD authentication frames were sent using an arbitration phase bit rate of 250Kbit/s and data phase bit-rates of 1 and 4Mbit/s while 250Kbit/s was used for the standard CAN authentication data as well as for the J1939 traffic. To ensure the timely reception of the authentication tag higher priority identifiers were selected for the frames carrying this information.

Sending the authentication tag along with the actual data in a single CAN-FD frame shows little to no effect on the busload as the higher data phase bit rate compensates for the longer message. As expected, the introduction of an additional authentication frame for each J1939 frame sent on the bus leads to an increased busload. The busload measured when our simulation runs without authentication support is at 19.39%. This value increases to 38.79% when adding the CAN-FD authentication frames, an increase which is consistent with a practically doubled traffic by the additional authentication frames. Sending the authentication tags as separate CAN frames has an even bigger impact on the busload increasing it to 58.18% with two CAN authentication frames and 77.57% when using three CAN authentication frames. The achieved busload can be improved by using a bigger bit rate.

For example, when using a 500Kbit/s bit rate the normal traffic load decreases to 9.7% while introducing the CAN-FD authentication frame leads to a busload of 18.45%. In this case CAN authentication leads to a 29.09% busload for two CAN frames and 38.79% for three CAN frames.

Scalability of busload results. As shown by the experimental results, the busloads resulted from the introduction of additional authentication frames show an increase proportional to the number of additional frames. This makes the results scalable allowing the estimation of the maximum feasible busload generated by normal unauthenticated traffic to allow the usage of the proposed mechanism depending on each approach of sending authentication tags. Since, when sending the authentication tag together with the actual message in the same CAN-FD frame with higher data rate results in a negligible effect on the busload there is practically no real limitation in the amount of J1939 traffic for this approach. When sending the tag in a separate frame the non-authenticated traffic is limited to 50%, 33.33% and 25% for an authentication tag sent trough a CAN-FD frame, two CAN frames and three CAN frames respectively.

We now discuss the impact of the authentication delays between regular J1939 frames and the additional CAN-FD authentication frames. Fortunately, as proved by the CANoe simulation, these delays are at around $460\mu s$ for a 24 byte at a data phase bit-rate of 1Mbit/s. This is small enough to avoid impeding with normal protocol operations.

Figures 13 and 14 show the delays and their distribution on the bus for 20 and 24 byte authentication frames when using either 1Mbit/s or 4Mbit/s data-rate. Delays obtained when using the same data-rate are similar as well as their statistical distribution. Thus reducing the data-field of the authentication frame by 4 bytes cuts the delays by 10–50 μs, this is a very small gain. By increasing the data-rate 4 times, from 1Mbit/s to 4Mbit/s, the delay between frames almost halves, from 420–480 μs to 260–280 μs. This is expected as the data-rate increases 4 times only during the data-field but not for the arbitration. The histogram distribution from Figure 14 shows a normal distrbution for the first 20.000 frames which clarifies that the recorded delays are the norm. On the 4Mbit/s variant the statistical distribution of the values is a bit sparse. This likely comes from the slightly lower traffic load obtained in this case. The statistical distribution does not affect the efficiency of the authentication mechanism since delay variations are in the order of a dozen microseconds, too small to be practically relevant.

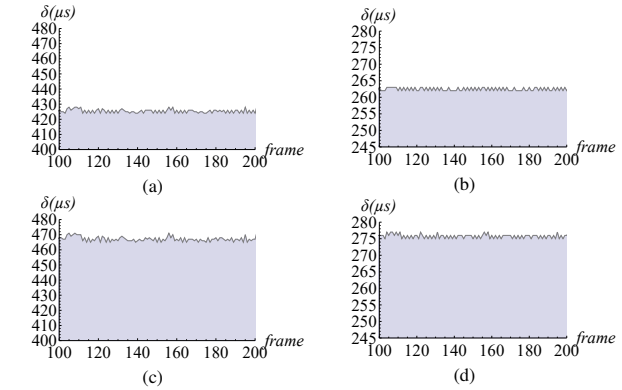

Fig. 13. Delays between regular frames (first 200) and the corresponding authentication frame with authentication field set to: 20 bytes and data-rate 1MBit/s (a), 20 bytes and data-rate 4MBit/s (b), 24 bytes and data-rate 1MBit/s (c), 24 bytes and data-rate 4MBit/s (d)

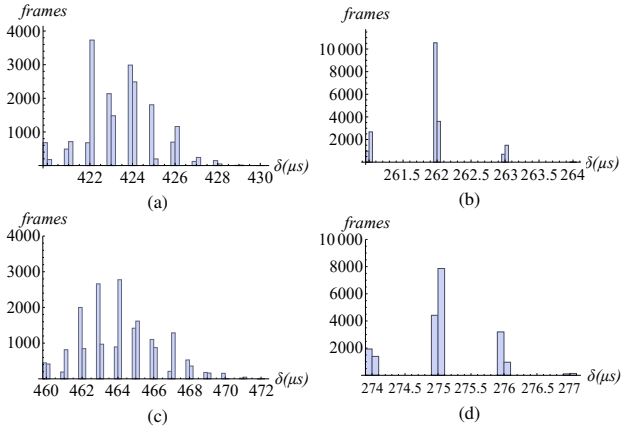

Fig. 14. Distribution of delays between regular frames (first 20.000) and the corresponding authentication frame with authentication field set to: 20 bytes and data-rate 1MBit/s (a), 20 bytes and data-rate 4MBit/s (b), 24 bytes and data-rate 1MBit/s (c), 24 bytes and data-rate 4MBit/s (d)

VI. Conclusion

We evaluated the shortcomings of the J1939 protocol specification and identified a series of issues that can be exploited to mount address impersonations and DoS attacks. By using an industry-standard CANoe simulation for the power-train of a commercial vehicle we successfully validated the mounted attacks (with one exception which revealed a behaviour non-compliant with the SAE J1939 protocol specification). As countermeasure for this, we proposed an authentication mechanism and evaluated its effect on the same network simulation by using CAN-FD frames to send the additional authentication data while also discussing the option of using CAN for this purpose. Our results show that the introduction of the proposed security mechanism does not have considerable effects on the bus communication reliability and CAN-FD proves to be an excellent layer for carrying the additional authentication data. As proved by our experiments, from a computational point of view, even low-end automotive-grade controllers are capable of handling symmetric cryptography. While low-end cores are limited to non-PKI operations, high-end cores can easily handle public-key cryptographic algorithms as demonstrated by the experimental section. The proposed solution tries to cope with both these scenarios. Since the PKI is quickly entering the automotive domain, as suggested by the introduction of public-key operations in the AUTOSAR specifications or in the requirements for V2X communications, future automotive-grade microcontrollers should easily handle all the operations required by our protocol.

Acknowledgment

This work was supported by a grant of the Romanian National Authority for Scientific Research and Innova-

tion, CNCS-UEFISCDI, project number PN-II-RU-TE-2014-4-1501 (2015-2017).

REFERENCES

[1] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway *et al.*, "Experimental security analysis of a modern automobile," in *2010 IEEE Symposium on Security and Privacy*. IEEE, 2010, pp. 447–462.

[2] S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, K. Koscher *et al.*, "Comprehensive experimental analyses of automotive attack surfaces." in *USENIX Security Symposium*, 2011.

[3] C. Miller and C. Valasek, "Adventures in automotive networks and control units," *DEF CON*, vol. 21, pp. 260–264, 2013.

[4] L. Dariz, M. Ruggeri, G. Costantino, and F. Martinelli, "A survey over low-level security issues in heavy duty vehicles," in *Automotive Cyber Security Conference*. ESCAR, 2016.

[5] Y. Burakova, B. Hass, L. Millar, and A. Weimerskirch, "Truck Hacking: An Experimental Analysis of the SAE J1939 Standard," in *10th USENIX Workshop on Offensive Technologies (WOOT 16)*, 2016.

[6] S. Mukherjee, H. Shirazi, I. Ray, J. Daily, and R. Gamble, "Practical DoS Attacks on Embedded Networks in Commercial Vehicles," in *Information Systems Security*. Springer, 2016, pp. 23–42.

[7] "J1939-21 – Data Link Layer," SAE International, Standard, Dec. 2010.

[8] O. Hartkopp, C. Reuber, and R. Schilling, "MaCAN-message authenticated CAN," in *10th Int. Conf. on Embedded Security in Cars*, 2012.

[9] A. Bruni, M. Sojka, F. Nielson, and H. R. Nielson, "Formal security analysis of the MaCAN protocol," in *Integrated Formal Methods*. Springer, 2014, pp. 241–255.

[10] Q. Wang and S. Sawhney, "Vecure: A practical security framework to protect the can bus of vehicles," in *Internet of Things (IOT), 2014 International Conference on the*. IEEE, 2014, pp. 13–18.

[11] A.-I. Radu and F. D. Garcia, "Leia: A lightweight authentication protocol for can," in *21st European Symposium on Research in Computer Security, ESORICS*. Springer, 2016, pp. 283–300.

[12] B. Groza and S. Murvay, "Efficient protocols for secure broadcast in controller area networks," *IEEE Trans. Ind. Inf.*, vol. 9, no. 4, pp. 2034–2042, Nov 2013.

[13] B. Groza, S. Murvay, A. V. Herrewege, and I. Verbauwhede, "LiBrA-CAN: Lightweight Broadcast Authentication for Controller Area Networks," *ACM Trans. Embed. Comput. Syst.*, vol. 16, no. 3, pp. 90:1–90:28, Apr. 2017.

[14] S. Jain and J. Guajardo, "Physical layer group key agreement for automotive controller area networks," in *Conference on Cryptographic Hardware and Embedded Systems*, 2016.

[15] A. Mueller and T. Lothspeich, "Plug-and-secure communication for CAN," *CAN Newsletter*, pp. 10–14, 2015.

[16] R. Kurachi, Y. Matsubara, H. Takada, N. Adachi, Y. Miyashita, and S. Horihata, "CaCAN - centralized authentication system in CAN (controller area network)," in *14th Int. Conf. on Embedded Security in Cars (ESCAR 2014)*, 2014.

[17] C.-W. Lin, Q. Zhu, C. Phung, and A. Sangiovanni-Vincentelli, "Security-aware mapping for CAN-based real-time distributed automotive systems," in *2013 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2013, pp. 115–121.

[18] C.-W. Lin, Q. Zhu, and A. Sangiovanni-Vincentelli, "Security-aware modeling and efficient mapping for CAN-based real-time distributed automotive systems," *IEEE Embed. Syst. Letters*, vol. 7, no. 1, pp. 11–14, 2015.

[19] S. Woo, H. J. Jo, I. S. Kim, and D. H. Lee, "A Practical Security Architecture for In-Vehicle CAN-FD," *IEEE Trans. Intell. Transp. Syst.*, vol. 17, no. 8, pp. 2248–2261, Aug 2016.

[20] S. Woo, H. J. Jo, and D. H. Lee, "A practical wireless attack on the connected car and security protocol for in-vehicle CAN," 2014.

[21] *CAN Specification Version 2.0.*, Robert BOSCH GmbH, 1991.

[22] *ISO 11898-1. Road vehicles - Controller area network (CAN) - Part 1: Controller area network data link layer and medium access control*, International Organization for Standardization, 2015.

[23] "J1939-71 – Vehicle Application Layer," SAE International, Standard, Mar. 2011.

[24] "J1939-81 – Network Management," SAE International, Standard, Apr. 2003.

[25] S. Cheshire, "IPv4 Address conflict detection," Internet Requests for Comments, RFC Editor, RFC 5227, July 2008. [Online]. Available: http://www.rfc-editor.org/rfc/rfc5227.txt

[26] *Specification of a Request Manager for SAE J1939*, AUTOSAR, 2015, release 4.2.2.

[27] *Specification of Network Management for SAE J1939*, AUTOSAR, 2015, release 4.2.2.

[28] *CAN interface for bodywork*, Scania CV AB, 2016, issue 1, Available: https://til.scania.com/groups/bwd/documents/bwm/mdaw/ntuw/~edisp/bwm_0001091_01.pdf.

[29] *Specification of Crypto Abstraction Library*, 4th ed., AUTOSAR, 2015.

[30] *Specification of Crypto Service Manager*, 4th ed., AUTOSAR, 2015.

[31] M. Raya and J.-P. Hubaux, "Securing vehicular ad hoc networks," *Journal of Computer Security*, vol. 15, no. 1, pp. 39–68, 2007.

[32] H. Hartenstein and L. Laberteaux, "A tutorial survey on vehicular ad hoc networks," *IEEE Com. Mag.*, vol. 46, no. 6, 2008.

[33] X. Lin, R. Lu, C. Zhang, H. Zhu, P.-H. Ho, and X. Shen, "Security in vehicular ad hoc networks," *IEEE Com. Mag.*, vol. 46, no. 4, 2008.

[34] W. Whyte, A. Weimerskirch, V. Kumar, and T. Hehn, "A security credential management system for v2v communications," in *Vehicular Networking Conference (VNC), 2013 IEEE*. IEEE, 2013, pp. 1–8.

[35] National Highway Traffic Safety Administration, Department of Transportation, "Vehicle-to-vehicle security credential management system; request for information," Office of the Federal Register, 2014.

[36] "wolfSSL Embedded SSL/TLS Library kernel description," https://www.wolfssl.com/, accessed: 2017-09-30.

[37] H. Zeltwanger, "Mapping of J1939 to CAN FD," *CAN Newsletter*, vol. 2, pp. 30–31, 2016.

[38] E. Barker and A. Roginsky, "Transitions: Recommendation for transitioning the use of cryptographic algorithms and key lengths," *NIST Special Publication*, vol. 800, p. 131A, 2011.

[39] W. Diffie and M. Hellman, "New directions in cryptography," *IEEE Trans. Inf. Theory*, vol. 22, no. 6, pp. 644–654, 1976.

[40] W. Diffie, P. C. Van Oorschot, and M. J. Wiener, "Authentication and authenticated key exchanges," *Designs, Codes and cryptography*, vol. 2, no. 2, pp. 107–125, 1992.

[41] T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *IEEE Trans. Inf. Theory*, vol. 31, no. 4, pp. 469–472, 1985.

[42] "CANoe tool," https://vector.com/vi_canoe_en.html.

[43] *General information on CAN*, Scania CV AB, 2016, issue 1, Available: https://til.scania.com/groups/bwd/documents/bwm/mdaw/ntux/~edisp/bwm_0001111_01.pdf.

Pal-Stefan Murvay is an assistant professor at Politehnica University of Timisoara (UPT). He graduated his B.Sc and M.Sc studies in 2008 and 2010 respectively and received his Ph.D. degree in 2014, all from UPT. He has a 9-year background as a software developer in the automotive industry as former employee of Continental Corporation (2005-2014). His current research interests are in the area of automotive security and works as a postdoctoral researcher in the CSEAMAN project.

Bogdan Groza is an associate professor at Politehnica University of Timisoara (UPT). He received his Dipl.Ing. and Ph.D. degree from UPT in 2004 and 2008 respectively. In 2016 he successfully defended his habilitation thesis having as core subject the design of cryptographic security for automotive embedded devices and networks. He has been actively involved inside UPT with the development of laboratories by Continental Automotive and Vector Informatik, two world-class manufacturers of automotive software. He currently leads the CSEAMAN project, a 2 years research program (2015-2017) in the area of automotive security.