Effective Intrusion Detection and Prevention for the Commercial Vehicle SAE J1939 CAN Bus

Camil Jichici, Bogdan Groza, Radu Ragobete, Pal-Stefan Murvay, Tudor Andreica

***Abstract*—Detecting and preventing intrusions on in-vehicle buses is a topic of great importance which may have an even greater significance in the context of commercial vehicles that are liable for the security of the demanding tasks they carry, passengers or goods not least. In this respect, the SAE J1939 protocol, which is a CAN based higher-layer protocol for commercial vehicles, requires special attention due to the existence of both specific procedures in the standard, e.g., address claims and multi-frame transmissions, as well as due to sharp specifications regarding the content of messages which may facilitate the deployment of a more targeted intrusion detection system. Needless to say, most of the research works on CAN intrusion detection are treating in-vehicle traffic as black-box with no concerns over the actual meaning of the frames content. In this context, we pursue the development of a targeted solution for J1939 buses. We collect real-world traffic from a commercial vehicle bus, compliant to the J1939 standard, and make a comprehensive analysis of its structure and content. This allows us to design an effective intrusion prevention system that detects and eliminates in real-time all frames that were manipulated by an adversary by overriding them with error flags. To prove the correctness of our approach, we present results with a proof-of-concept implementation on high-end automotive-grade controllers.**

***Keywords*-CAN bus, Intrusion Detection and Prevention, J1939**

I. Introduction and motivation

As the number of Electronic Control Units (ECUs) employed inside vehicles has drastically increased in the past decades, so did the complexity of in-vehicle networks. Currently, the communication between in-vehicle ECUs is mediated by several communication interfaces such as the classical Controller Area Network (CAN) or the more recent 100Base-T1 Ethernet which is capable of much higher data-rates. But despite recent advances, CAN is by far the most commonly used communication layer and provides a very good cost-performance ratio for most applications. A contemporary extension of it, i.e., CAN-FD, allows much higher data-rates and larger packets which sets room for the endurance of CAN even in the decades that follow.

However, a common issue of all in-vehicle communication interfaces is their lack of security. This led to several reported attacks, e.g., [1], [2], [3], and such attacks may have life-threatening consequences for vehicle passengers or for traffic participants. Unsurprisingly, most of the reported attacks so far on in-vehicle buses are due to the insecurity of the CAN bus and many of them are mounted from the On Board Diagnostic (OBD) port which easily exposes at least part of the in-vehicle network. This is graphically suggested in Figure 1 which gives a brief visual representation of the setup that we address: an adversary that plugs on the CAN bus of a commercial vehicle network, being able to read or write messages on the bus. In-vehicle networks are not always flat. Usually, multiple ECUs, that implement related functionalities, are grouped on the same bus and multiple buses are linked by gateways in a hierarchical topology. However, the network inside the heavy-duty vehicle that we study is flat as indicated by the traffic collected from the OBD port which contains information that is not related to vehicle diagnosis, proving that there is no gateway to filter incoming traffic. In case of hierarchical architectures, the solution that we design can be applied to each sub-bus, while gateway ECUs that usually have higher computational resources can take care of the more demanding functionality in our protocol, i.e., detecting the intrusions and destroying the adversarial frames with error flags. Numerous research works emerged in response to these attacks (we discuss some of them later) but most of the research efforts so far targeted CAN in the context of passenger cars. The commercial vehicle sector brings on specific challenges due to both its liability for the security of multiple passenger or goods, as well as due to its technical particularities.

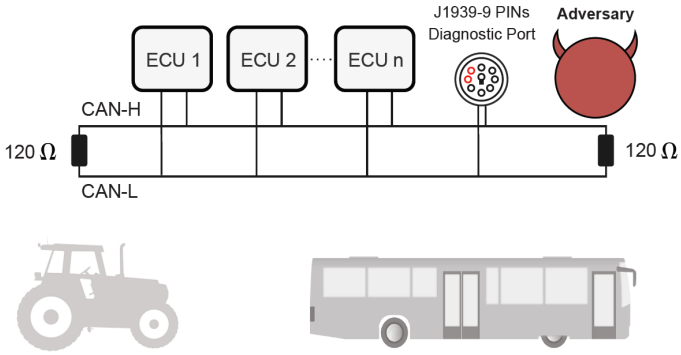


Fig. 1. Targeted setup: a typical commercial vehicle J1939 CAN bus (present inside tractors, buses, etc.)

The SAE J1939 is a higher layer protocol built on top of the CAN specification for commercial vehicles. SAE J1939 standardizes CAN-based communication in commercial vehicles adding message formatting and network management mechanisms. Of course, similar to the case of standard CAN, security was not considered when designing SAE J1939 and

Camil Jichici, Bogdan Groza, Pal-Stefan Murvay, Tudor Andreica are with the Faculty of Automatics and Computers, Politehnica University of Timisoara, Romania, Radu Ragobete is with Continental Automotive Timisoara, Romania, Email: {camil.jichici, bogdan.groza, pal-stefan.murvay, tudor.andreica}@aut.upt.ro, radu.ragobete@continental-corporation.com

this enables a series of attacks targeting specific J1939 features [4]. However, the comprehensive specifications of J1939 buses can be exploited in a positive manner since they set room for designing specialized procedures to assure the security of the bus. Specific details such as node addresses and message structure (or content) can be more thoughtfully examined so that intrusions can be more efficiently detected. Intrusion detection systems (IDS) are components that monitor networks for malicious traffic, some of them also having the ability to respond to such intrusions by actively blocking them. These later systems are typically referred to as intrusion prevention systems. The system designed in our work is mixed allowing the detection of malicious frames by their content and encrypted addresses and also their destruction by error flags if frame classification can be done in real-time by a high-end ECU (this later functionality is demonstrated in our experiments by using a high-end automotive-grade ARM Cortex R5 platform). While most of the research works on CAN intrusion detection addressed in-vehicle traffic in a black-box manner, the J1939 standard specifications allow for a crisper view on protocol details. Our work goes in this vein as we develop an effective intrusion prevention system for J1939 buses that is able to detect intrusions and react in real-time by efficiently removing adversarial frames from the bus. The contributions of our work can be briefly summarized as follows:

1) we design an intrusion detection and prevention system that specifically targets J1939 specifications (so far, an intrusion detection and prevention system tailored for J1939 buses is missing from the research literature),
2) a special feature of the proposed system is the ability to destroy intrusion frames immediately, without the need of specialized hardware by using the input capture unit (ICU) to bypass CAN buffers and interpret CAN frames right before the acknowledgment bit is placed,
3) we use a realistic testbed with traffic collected from a real-world J1939 heavy-duty vehicle that allows us to make an accurate assessment of the intrusion detection rates of the proposed system,
4) to assess the applicability of the proposed solution, we provide computational results on four representative automotive-grade platforms: a high-end 32-bit S6J32GEKSN from the Cypress's Traveo family, two high-end Infineon Tricore platforms and a low-end Freescale S12XF512 controller,
5) the frame destruction capabilities in real-time are demonstrated on attacks performed in a laboratory setup with the more powerful Traveo S6J32GEKSN development board using traffic collected by us from a real-world J1939 heavy-duty vehicle.

The remainder of this paper is organized as follows. We briefly discuss the related work in Section I.A) focusing mostly on existing intrusion detection systems for the CAN bus. In Section II we discuss some background on CAN and the J1939 specifications. Section III presents our data collection procedure and a careful analysis of the collected in-vehicle traffic. In Section IV we present the design of the proposed solution and in Section V we discuss experiments. Finally, Section VI holds the conclusion of our work.

A. Related work

J1939 security. Several lines of work employed different approaches for the security analysis of the SAE J1939 protocol. Burakova et al. [5] demonstrated that injection and replay attacks previously reported in standard CAN networks are also effective for attacking J1939 based communication in commercial vehicles. Mukherjee et al. [6] are the first to look at attacks specific to the SAE J1939 protocol and illustrate a DoS attack on the transport protocol employed for transmitting multi-frame messages. An in-depth analysis of the J1939 protocol specification, presented in [4], reveals a number of attacks involving the address claim procedure, and transport protocol mechanisms capable of constraining the communication capability of compliant nodes.

Steps were also taken towards securing J1939 communication. An authentication protocol for J1939 is designed and evaluated in [4]. The downside of this proposal comes from the high communication overhead which would make it more suitable for CAN-FD communication. The use of encrypted J1939 communication was also evaluated [7] but only in the context of diagnostic messages. Another line of work [8] investigates the use of precedence graph-based anomaly detection for detecting malicious messages in J1939 traffic while the use of machine learning is applied for intrusion detection in SAE J1939 networks in [9].

CAN intrusion detection. Numerous security solutions for CAN communication were proposed in the recent years, an overview can be found in [10]. The industry is aware of the security shortcomings of CAN buses and recent standards, such as the AUTOSAR standard for Secure On-board Communication [11], try to alleviate this problem by adding cryptographic payloads, i.e., a truncated Message Authentication Code (MAC), to each frame. This type of cryptographic protection can be traced back to earlier papers such as [12], [13], [14]. However, in case of J1939 specifications, the entire datafield of most frames is already fully loaded with signals and there is no space left to introduce security elements such as truncated MACs. For this reason, to eliminate the need for security bits inside frames, our work takes a different approach by using encrypted addresses which will hinder an adversary in injecting correct IDs in the network and by using encrypted payloads which, due to the avalanche effect of block ciphers, set room for effective intrusion detection by range checks.

The design of IDS is a specific area of interest which has recently emerged in a number of papers. A comprehensive overview of IDS implementation on the CAN bus is provided in [15]. The authors investigated several IDS proposals from the literature taking into consideration important aspects in the detection approach, the deployment strategies, the attack technique and technical challenges. Neural networks based intrusion detection systems are proposed in [16] and [17]. A change detection approach for Controller Area Network built on a density ratio estimation method is proposed in [18].

Further intrusion detection mechanisms employing deep learning techniques are presented in [19] and [20]. In the

latter, the authors propose the extension of on-board detection mechanisms with cloud-based computational offloading in order to overcome the high processing demands. Wang et al. presented a real-time anomaly detection framework in [21], making use of hierarchical temporal memory (HTM) learning algorithm.

An intrusion detection methodology based on remote frames is discussed in [22]. The detection method examines the response offset ratio and time interval of the remote frames responses to detect intrusions. A dynamic ensemble selection system (DESS) for anomaly detection in metro trains braking control system is described in [23]. The proposed anomaly detector relies on two-class and one-class classifiers and is able to detect, without expert knowledge, known and unknown fault types.

Choi et al. proposed VoltageIDS in [24] which is based on the inimitable characteristics of electrical CAN signals. VoltageIDS is easy to install and its feasibility was validated on real vehicles being in motion. Another work that uses physical characteristics of voltage CAN signals for intrusion detection is [25]. The proposed approach uses Local Outlier Factor (LOF) algorithm which efficiently detects intrusions and has a low false detection rate. The use of Bloom filtering for intrusion detections on CAN-bus was examined in [26]. Intrusion detection mechanisms based on entropy analysis are suggested in [27], [28] and [29]. An optimized information entropy based IDS is proposed in [30] and a graph-based approach that follows the sequences of messages is suggested in [31].

A light-weight IDS for CAN-bus relying on the analysis of time intervals of messages is discussed in [32]. The work proved that the timestamps of the messages can be efficiently used to detect messages injected on the CAN bus. A performance comparison of four light-weight intrusion detection methods for in-vehicle networks is performed by the authors of [33]. The compared approaches detect intrusions based on: information entropy [27], clock skews [34], ID sequence [35] and throughput [36].

II. Brief overview of CAN and J1939 specifications

This section gives a brief overview on CAN and the SAE J1939 standard arrangements, outlining details regarding the frame identifier, address claiming procedure and multi-frame transmissions which are specific to J1939 implementations.

A. CAN overview

The CAN protocol was introduced by BOSCH [37] in the 80s. Subsequently, the specification was standardized as the ISO 11898 standard which covers details of the various protocol layers, e.g., [38] for generic data-link and physical layer and [39] for high-speed CAN medium access unit. At the physical layer CAN is implemented as a two-wire (CAN-High, CAN-Low) bus which employs differential signaling. The already introduced Figure 1 illustrates a two-wire CAN bus which enables communication between ECUs inside a commercial vehicle.

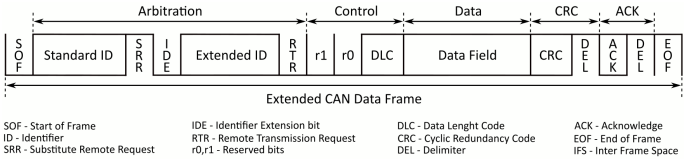


Fig. 2. Extended CAN Data Frame Format

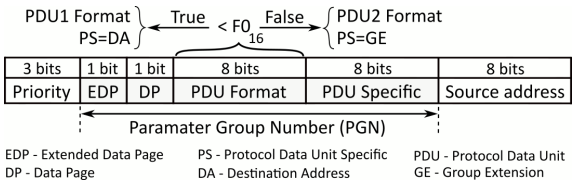


Fig. 3. Structure of the J1939 Message Identifier

Bit rates of up to 1 Mbit/s can be achieved using high-speed CAN. Figure 2 shows the structure of the extended CAN data frame. Each CAN frame begins with a dominant SOF bit and ends with a recessive EOF bit. The arbitration field holds the frame identifier ID, which is 11 bit long in standard frames and 29 bits in extended frames, along with the SRR, IDE and RTR bits. This field is employed as part of the arbitration mechanism used to assure a collision-free bus access. In particular, in case of commercial vehicles, it denotes the message content according to J1939 standard specification. The control field consists of two reserved bits and the DLC which indicates the number of bytes that will be enclosed in the data field. The frame data field can hold up to 8 bytes. A check regarding the correctness of the frame is done through a 15-bit CRC which ends with a recessive CRC delimiter bit. The sender transmits a recessive bit in the ACK slot which will be overwritten by the receiver with a dominant (0) bit in case of correct reception.

CAN with Flexible Data-Rate (CAN-FD), an extension of the standard CAN specification was more recently developed to cope with the data throughput limitations given by the maximum 8 byte payload and 1Mbit/s bit rate. This newer extension, CAN-FD, supports payloads of up to 64 bytes and higher bit rates for the data field (i.e. currently available transceivers are capable of up to 8Mbit/s rates). The commercial vehicle that we analyze has however a regular CAN bus with 8 byte data-frames.

B. Particularities of SAE J1939

As stated, the commercial vehicles sector (trucks, buses, tractors, mobile cranes, excavators etc.) of the automotive industry employs the SAE J1939 standardization for the CAN protocol. At the physical layer, SAE J1939 uses the ISO 11998 standard specification with some minimal additions such as limitations regarding bit rates. Other specific features are defined at the upper layers of the communication stack such as data link layer features which are specified in SAE J1939-21 [40].

Structure of the ID according to J1939 specifications. One essential particularity of the SAE J1939 specification

is the use of extended CAN frames only (i.e. frames with identifiers of 29 bits rather than 11 bits). A specific format is defined for the ID field as depicted in Figure 3. The J1939 message identifier is divided into six fields: priority, extended data page (EDP), data page (DP), protocol data unit format (PF), protocol data unit specific (PS) and source address. Four of these fields, i.e., EDP, DP, PF, PS, are used to form the Parameter Group Number (PGN) which uniquely identifies a parameter group (PG) indicating that a set of specific parameters, e.g., data, acknowledgments etc., are contained into a J1939 frame. The J1939-71 white paper [41], defines a comprehensive set of standardized J1939 frames, along with the corresponding PG values, covering the needs of most standard ECU types found in commercial vehicles while also allowing for OEM specific frames. The data field of a J1939 frame (i.e., the signals that are included in each data frame) can be decoded based on the standard specifications as long as the particular frame type is not OEM specific. The J1939 Digital Annex document [42] complements the standard and presents specific information in an easily accessible form. Moreover, this document includes the preferred addresses for ECUs by all J1939 targeted areas, e.g., agricultural and forestry equipment, on-highway equipment, etc. The PDU format classification is done according to the value of the PDU format field. In this context, there are two types of PDU formats labeled as PDU1 and PDU2, respectively. The PDU1 format is used when the PF field value is below 240, while the PDU2 format is recognized when the PF value is between 240 and 255. The most significant difference between these formats is related to the interpretation of the PS field which contains a destination address (DA) for PDU1 and a group extension (GE) for PDU2.

Node address and address claims. Node addresses, used as frame source and destination, represent unique identifiers assigned to each node within a J1939 network. For broadcast addressing (i.e., transmission to all nodes), the value 0xFF will be employed as the DA. Only nodes that own an unique address are permitted to communicate. According to the J1939 network management specification defined in SAE J1939-81 [43], nodes can secure an address through the address claiming procedure which takes place before initiating the communication within the J1939 network. Usually, this procedure takes place during the system power-up sequence. Nodes that are later added to the network or initialized upon request will undergo the address claiming procedure once started. According to [43], the CAN frame that uses PGN (60928) is associated with the address claiming procedure and is transmitted on request. Consequently, this request (a specific frame with PGN 59904) can be sent by any node from the network. Each request can be sent to the global address (DA=0xFF) or to a specific destination address. A node should claim his own address by sending a frame, with PGN 60928, containing its NAME and requested address, before sending any kind of message. If this request is addressed globally, then all ECUs within the network should respond with a message with the Address Claiming PGN (60928), its particular NAME and address, in case the node has already claimed an address. If a node fails to successfully claim his address, it specifies the null address (254) in the address field to indicate failure

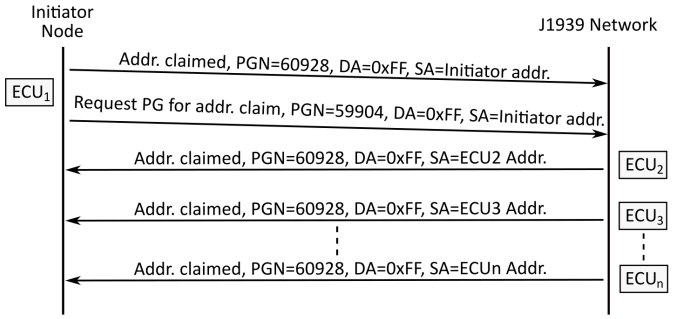


Fig. 4. Frames exchange sequence during the J1939 address claiming procedure

in claiming an address. The NAME is a parameter, unique to each node, which reveals information about the ECU's functions, manufacturer code, identify number and industry sector. An example of address claiming procedure is depicted in Figure 4. In this example a *Initiator Node* claims his own address and then sends a broadcast request for a PGN that is associated with Address Claim by using the global address. As a response, all ECUs within the J1939 network, that already claimed an address, send an Address Claim message indicating their own source address.

Multi-frame transmissions. An additional feature included by the J1939 protocol, is the use of transport protocols that allows the transmission of payloads of up to 1785 bytes through multi-frame messages [40]. Since a CAN frame can carry up to 8 data bytes, messages longer than 8 bytes can only be sent by fragmenting the data in several CAN frames. This is possible due to the existence of several specific J1939 messages (Table I) which are part of the Transport Protocol Connection Management (TP.CM) and Transport Protocol Data Transfer (TP.DT). TP.CM messages are used in order to initiate connection and control the frame exchange between the sender and receiver nodes while TP.DT messages are employed for the actual data transfer. These can be used for two types of data transfer:

1) **Broadcast Data Transfer** as outlined in Figure 5 (i). Here, in the first step the initiator node sends a BAM (Broadcast Announce Message) which informs the network that a multi-packet message will follow. Next, the same node transmits the DT (Data Transfer) messages. The message exchange is controlled only by sender node.
2) **Connection Mode Data Transfer** as depicted in Figure 5 (ii). Here a connection between the sender and receiver nodes is established through an RTS (Request to Send) - CTS (Clear to Send) message exchange, followed by the DT messages. The end of a multi-packet message transmission is indicated by the receiver node through an End of Message Acknowledgment (EndOfMsgACK) message. The connection between initiator and receiver node can be interrupted at any moment by a specific message, i.e. Connection Abort (Conn_Abort) message.

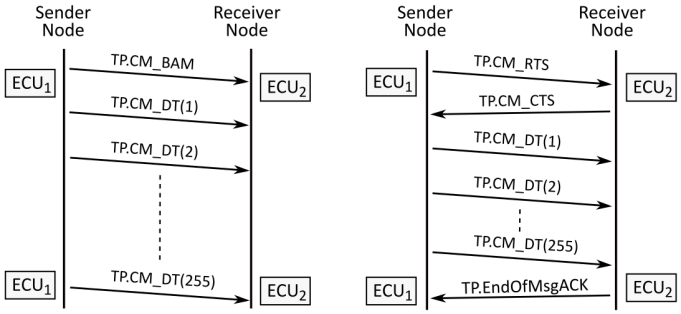


Fig. 5. Multi-frame transmissions: Broadcast Data Transfer (left) and Connection Mode Data Transfer (right)

TABLE I
J1939 SPECIFIC TRANSPORT PROTOCOL MESSAGES

Message	PF	PS	Description
TP.CM_BAM	236	FF	broadcast message which inform the nodes about a specific PG and a fixed number of data bytes that will enclosed within a multi data packet message
TP.CM_RTS	236	DA	is used by a node which intends to initiate a connection with another node
TP.CM_CTS	236	DA	is sent by receiver node being used as a response to TP.CM_RTS message, and inform the connection initiator node that it is ready to receive a fixed number of data bytes
TP.CM_EndOfMsgACK	236	DA	is used by the receiver node in order to inform the sender node that all data bytes are received successfully
TP.Conn_Abort	236	DA	can be employed by sender or receiver in order to close a connection before the data transmission is finished
TP.DT	235	DA	is used by the initiator node in order to transmit the data packets

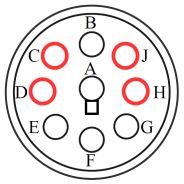
III. Data collection and analysis

In this section, we present the data collection methodology and then we examine the collected CAN traffic to outline specific details from it.

A. Data retrieval from the J1939 9-PIN Diagnostic Port

As a practical instantiation of J1939 traffic, we decided to collect traffic from a modern agricultural vehicle from a reputable manufacturer. For confidentiality reasons, we do not specify the identity of the manufacturer but the collected data perfectly match the J1939 specifications.

To avoid unnecessary complications, first we performed a quick examination with the help of an oscilloscope to determine if CAN traffic is exposed on the 9-PIN diagnostic port specific to J1939 implementations [44]. Figure 6 depicts the schematic of the diagnostic port connector according to J1939 specifications indicating the pins employed for CAN communication. We analyzed the presence of traffic on both CAN channels (pairs C-D and H-J) and we determined that traffic is available only on the the main channel (pin pair C-D) which used a bit rate of 250 Kbps. No traffic was observed on the second CAN channel (pin pair H-J) which is reserved for OEM (original equipment manufacturer) specific implementations. After interfacing with the J1939 diagnostics connector, we started to log the CAN bus traffic for about half an hour. During this period, various generic driving-related



C- CAN_H
D- CAN_L
H- Reserved for OEM or implement CAN_H
J- Reserved for OEM or implement CAN_L

Fig. 6. The 9 PIN Diagnostic Port of J1939

Fig. 7. Experimental setup employed for data collection

and tractor-specific actions were performed such as: driving forward/backwards, manipulating the bucket, etc. All these were done in an attempt to trigger as much events as possible which will result in a more complete set of CAN data frames.

Figure 7 depicts our data acquisition setup inside the tractor which consists in a Vector VN1640 USB-to-CAN interface, the CAN cable and the logging application (based on the Vector XL Driver Library) running on a laptop. The VN1640 is a device produced by Vector Informatik, a major provider for automotive networking solutions, and is part of the VN1600 family. This device allows interfacing computers with CAN, LIN, K-Line, J1708 communication lines and is supported by several traffic analysis and testing applications such as CANoe and CANape. The XL Driver Library provides a freely available API (Application Programming Interface) for interfacing with Vector devices allowing basic functionalities such as baud rate configuration, channel configuration, frame transmission/reception etc. We did not inject any adversarial frames on the vehicle bus in order to avoid inflicting any sort of damage to the vehicle. For each received frame, we record a time-stamp (in nanoseconds), the ID, the DLC and the actual data field and store these in our experimental trace. More details on the collected data follow.

B. Analysis of J1939 specific elements from traffic

The collected CAN traffic contained 51 IDs, out of which only 41 occur at runtime (the other 10 IDs occur only at startup). These IDs were interpreted by us according to the J1939 ID breakdown. For brevity, we present the detailed breakdown in Table VI which is deferred to the Appendix of this paper. Based on the ID analysis, we extracted distinct source addresses to reveal the ECUs present in the CAN network that are exposed to the outside through the diagnostic

port. We found that the collected traffic contained three different SAs as follows: 0x00, 0x03 and 0x21. These are specified in the J1939 Digital Annex document [42]. Based on this document, we identified the ECUs that are present in the CAN network of the test vehicle as responsible for the following tasks: ECM (Engine Control Module - 0x00), TCM (Transmission Control Module - 0x03) and BCM (Body Control Module - 0x21). A brief description of the identified ECUs may be in order:

1) *ECM (Engine Control Module)* - is one of the most important ECUs inside vehicles, it retrieves values from several engine sensors and interprets them in order to control the engine actuators through the following parameters: air-fuel ratio, fuel injection, ignition and variable valve timing for achieving the maximum engine performance.
2) *BCM (Body Control Module)* - is the central ECU that covers several specific body functions such as: exterior and interior lighting, power windows, seat position, climate control and central locking.
3) *Transmission Control Module (TCM)* - is responsible for automatic transmission and has as inputs values from sensors such as: turbine speed, wheel speed, throttle position, etc. The TCU uses those inputs in order to adjust the gears in order to achieve an efficient performance. This is done through several outputs such as: torque converter, clutch solenoid, shift lock, etc.

The recorded trace also contains multi-frame messages that are J1939 specific and we identified that, as expected, the Broadcast Data Transfer protocol was employed. To avoid overloading the main body of the work, we present some details regarding the collected IDs and their structure in Tables VI and VII from Appendix A.

C. A quantitative analysis of network traffic

The previous analysis outlined elements at a logical level according to the J1939 specifications which are helpful for traffic reconstruction. We now focus on a quantitative analysis of the recorded traffic that will allow us to set some limits on the intrusion detection and prevention system.

Generally, in-vehicle ECUs are broadcasting frames with specific IDs at fixed cycle times. The J1939 deployment that we study is no exception to this. By careful examination of the traffic we determined that the IDs are broadcast at 20, 25, 50, 100, 500 ms. There are also several slower ID that are broadcast at 1s. Multi-frame IDs arrive rarely, e.g., at 5s, and as expected are followed by a burst of frames. The plots in Figure 8 show that the arrival time is generally stable with very small fluctuations for IDs arriving at: (i) 20, (ii) 25, (iii) 50, (iv) 100 and (v) 500 ms. The multi-frame transmission exhibits a different behavior with a slow cycle time of $5s$ followed by quick bursts as can be seen in part (vi) of Figure 8. Nonetheless, it is relevant to point on the delays between frames in the entire trace since the security mechanism has to cope with this constraint. Figure 9 shows the delays between the reception of two consecutive frames for the first 1000 frames in the trace (to avoid overloading the figure). The right part of the Figure shows their histogram distribution for the

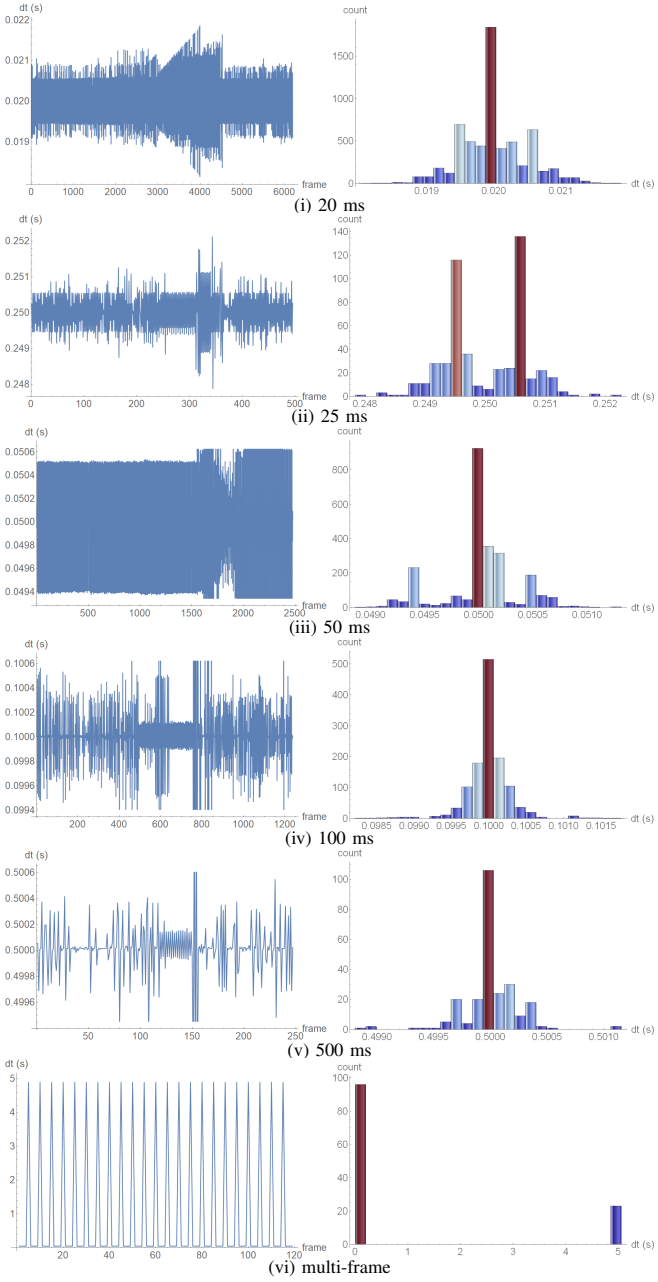


Fig. 8. Recorded inter-frame delays and their histogram distribution for IDs arriving at: 20, 25, 50, 100, 500 ms and a multi-frame transmission

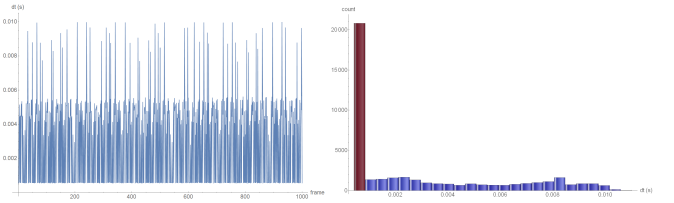
entire trace. The delays can get as lower as $500\mu s$ which is roughly the time spent on the bus by a CAN frame at 250 Kbps. In principle, this means that any employed security mechanism has to be executed in at most $500\mu s$ in order to set the ECU ready for the next frame.

IV. Proposed Mitigation Technique

In this section we present the complete solution that we envision for protecting J1939 communications.

A. Adversary model

The adversarial actions typically considered in the development of intrusion detection systems for CAN buses include



(i) (ii)

Fig. 9. Delays between frames for the first 1000 frames and histogram distribution of the delays for the entire trace

replay, modification or DoS (Denial of Service) attacks [26], [31], [45]. We note that while some papers include spoofing attacks in the adversary model, this type of attack refers to the replay or modification of frames with known IDs. Also, some works have targeted fuzzying attacks [22] in which IDs unknown to the network are introduced on the bus. Such attacks will be trivial to detect by simply checking if the ID is known to the network. While we do not explicitly address fuzzying attacks, they will be immediately detected as we verify that the ID of each frame is part of the known identifiers.

In the adversary model from this work, we consider both replay and modification attacks. Detecting replays is facilitated by the use of encrypted addresses while modifications are detected by range checks taking benefit of the avalanche effect in the encrypted data-field. We do not specifically include DoS attacks on the bus, on the CAN frame format or on the error confinement mechanism since due to the nature of the CAN bus (ID oriented arbitration) such attacks cannot be stopped. An adversary can always destroy frames and write high priority IDs to flood the bus as proved by related works, e.g., [46], [6]. DoS attacks will not be possible to prevent by any other previously explored IDS in the literature. Of course, detecting DoS attacks by flooding would be trivial by simply observing the busload but the attacks cannot be stopped due to the nature of CAN buses. Targeted DoS attacks, in which frame content is manipulated to cause transmission errors and put nodes into a bus-off state as demonstrated in [46], are much harder to detect and require modification in the CAN error handling mechanism that cannot be addressed by an intrusion detection system.

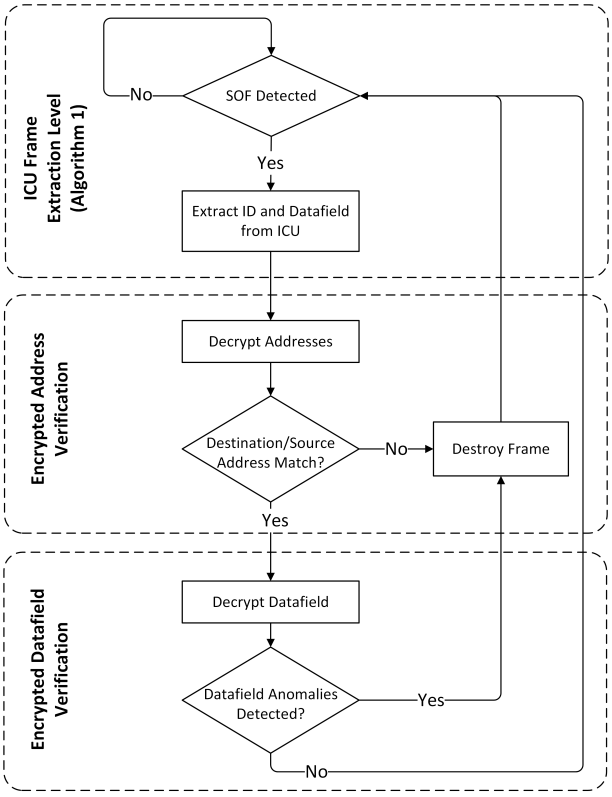


Fig. 10. The two-layer intrusion detection: encrypted address verification and field anomaly checking

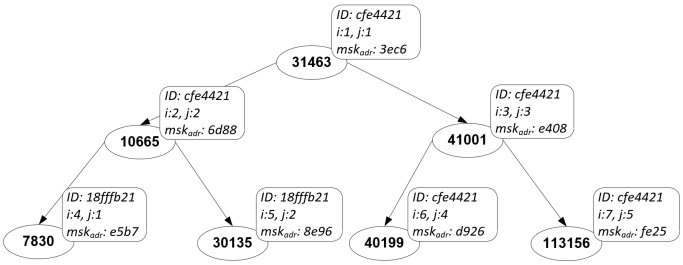


Fig. 11. Generating the ID masks by means of an ordered binary tree (to assure uniqueness of each address)

B. Overview of the proposed solution

We imagine a two-layer intrusion prevention system that is responsible for checking the validity of the source and destination addresses as well as for detecting anomalies in the data-field. To achieve this, we re-map source and destination addresses to encrypted values that are known only to legitimate nodes from the network. Further, the data-field can be encrypted as well since this (along with the encrypted parts of the ID) will hinder an adversary from deducing the structure of the data-field. Thus we argue that identifier encryption along with data-field encryption will confuse the adversary on the exact meaning of the frame (despite existing specifications in the J1939 standard) and will make malicious manipulations of the datafield easier to spot by the system. Figure 10 provides an overview of the two layer intrusion prevention mechanism that we propose. According to the schematic from Figure 10, whenever a start of frame is detected, i.e., a transition from the recessive state to the dominant state, the ICU begins constructing the frame based on Algorithm 1. This is needed in order to extract frame content before the frame is available in the CAN buffer since at that point the frame has already become available to network nodes and it can no longer be destroyed with an error flag. Then the algorithm enters the two stages of intrusion detection which consists in two types of mechanisms, the use of *encrypted addresses* and *datafield encryption* which are presented next. Once an intrusion is detected, adversarial frames are destroyed with error flags.

Encrypted addressees are used for replacing the regular address of the sender/receiver nodes with an address that is secured by a cryptographic one-way function (AES encryption). To prevent an adversary for re-using a disclosed address, the node address is re-mapped periodically based on an address

mapping matrix. Securing the ID field of the CAN frame is inline with recent works proposing similar solutions [47], [48], [49], [50]. We emphasize however that distinct to these works which intend to preserve the designated priority of the ID, this is not the case in our proposal which is solely concerned to encrypting the addresses of the source and destination nodes (the J1939 encodes the priority only in the first three bits of the ID which are unaltered in our protocol).

By properly encrypting the sender and destination addresses in the ID field, each being 8 bits long, we achieve 16 bits of security. This can be of course extended to 24 bits of security to comply with the AUTOSAR specifications [11] by encrypting the PDU format of the frame. We believe however that leaving at least part of the ID field unencrypted will be preferable for ID filtering mechanism, a reason for which we keep with the 16 bit security level which will give only a small chance to the adversary to successfully inject a message on the bus. Of course, in order for the injection to be successful, manipulations of the data-field must go undetected which will further lower the adversary success rate.

Data-field encryption and authentication is an additional protection layer in which by using a fast encryption mechanism with a lightweight block cipher we conceal the true content of the frame and set room for an avalanche effect in case of single bit manipulations that will be easily observed by proper range checks. By employing SPECK [51], one of the fastest existing block ciphers, decryption can be performed fast enough by the high-end controllers (in a few micro-seconds) allowing the destruction of malicious frames in real-time as we later show in the experiments. Encrypted traffic analysis may still be an option for well-determined adversaries, however, since both the identifiers and data-fields are encrypted and thus pseudorandom, it should be very difficult to extract any meaningful information from the encrypted frames. Due to space constraints, a more in-depth analysis of such an attack scenario is out of reach for the current communication.

Key management. The mechanism that we propose requires each CAN node to be in possession of a symmetric secret key. In our implementation, this key was a randomly generated value hardcoded on each node. Secure key establishment is a distinct subject that has been addressed by several recent works and it would be out of scope for us to design procedures for this purpose. For this reason, we only point out to some recent proposals addressing such issues. The work in [52] proposes the use of the physical layer to securely exchange keys between CAN nodes and allows the creation of group of nodes that use the same keys. Group keying with implicit certification based on the elliptical curve Diffie-Hellman protocol has been recently proposed in [53]. The use of elliptic curves for key exchange on the CAN bus is also considered in [54] and [55].

C. Unique encrypted addresses by ordered binary trees

One specific problem when generating random addresses that re-map the original ones, is the possibility of the addresses to collide. That is, if each pair (DA, SA) is replaced by a random 16 bit value, based on the birthday paradox we have a probability of collision of 50% at roughly $\sqrt{65536} = 256$ values. This probability is high and such an event likely to occur which will result in IDs being mismatched by legitimate nodes. To prevent this, we map the IDs based on an ordered binary tree where only unique values are kept.

Figure 11 shows how we construct the binary tree. Each node of the tree is inserted based on its value that is computed by XOR-ing the last 16 bits of the ID (which are the destination and sender addresses) with adress mask $\mathsf{msk}_{\mathsf{adr}}$. For example, the first node in the tree has an associated value of 31463 which is the result of XOR-ing the last 16 bits of the original ID, i.e., hexvalue CFE4421, with $\mathsf{msk}_{\mathsf{adr}} = 3\text{EC6}$, i.e., $4421 \oplus 3\text{EC6} = 7\text{AE7} = 31463_{10}$. Of course, in this case the new sender address is E7 while the new destination address 7A. The binary tree is ordered by this resulting value in order to easily check for collisions in the resulting addresses. The counters i and j are incremented each time a new value is generated, counter i is the global counter for ID generation while counter j is the local counter specifying the index of the value for the current ID. In case of a collision, the counter i is incremented and a new value is generated. Such an event will not occur too often but it is a possibility that needs to be covered.

Updating key masks in the sorted binary tree raises both computational and memory concerns. We now formalize these constraints and discuss trade-offs. We consider the set of identifier-cycle pairs that are defined on the CAN network $\{(id_1, \delta_1), (id_2, \delta_2), ..., (id_n, \delta_n)\}$, the life-time of the key tree Δ and the unicity interval of the keys δ. The later parameter is the time interval for which all the encrypted addresses are new. Ideally $\delta = \Delta$ such that all addresses are distinct for the entire lifetime of the tree but this may require too much memory and computational power that when not available for all nodes, a $\delta < \Delta$, may be selected. Since most of the traffic on CAN buses is cyclic, we can immediately define the ratio of unique encrypted addresses as $\rho = \delta\Delta^{-1}$. Based on these, the number of encryption masks λ_i generated for each $id_i, i = 1..n$ and the total number of encryption masks Λ are: $\lambda_{i\in\{1..n\}} = \rho\dfrac{\Delta}{\delta_i} = \dfrac{\delta}{\delta_i}$ and $\Lambda = \sum_{i=1}^{n} \lambda_i$ respectively. Due to possible collisions, the number of encryption masks is slightly larger than Λ leading to an average computational time of

$$\mathbb{T}_{\mathsf{comp}} = \left(\sum_{i=1}^{n} \frac{2^{16}}{2^{16} - i} \right) \times t_{crypt}.$$

Here t_{crypt} is the time required to generate one encrypted address. Since to encrypt the address we simply use 16 bit address masks and one AES computation generates 128 random bits, $t_{crypt} = t_{AES}/8$ where t_{AES} is the time for one AES encryption which will generate 8 encryption masks. For our practical needs, given that the number of IDs is low, i.e., there are usually less than 100 IDs on real-world buses while our vehicle has 41, $\mathbb{T}_{\mathsf{comp}}$ can be safely approximated as $\mathbb{T}_{\mathsf{comp}} \approx \Lambda \times t_{crypt}$. This estimation is very close to the actual number since on average to generate 50-100 unique encrypted addresses in the 2^{16} address space will require only 1-2 additional encryptions with a probability of 0 collisions

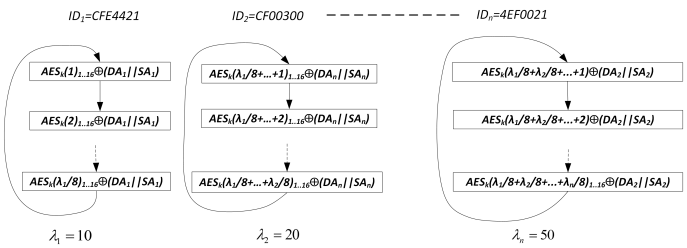


Fig. 12. Sender and destination address re-mapping by counter mode encryption and circular lists

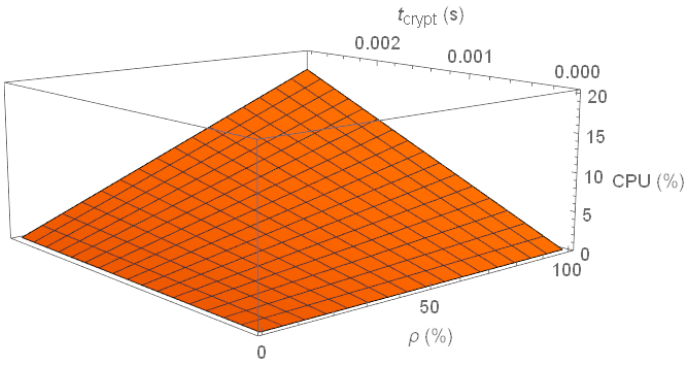


Fig. 13. CPU load during address generation in relation to encryption time t_{crypt} and coverage ρ)

equal to $\frac{2^{16}!}{(2^{16}-n)!2^{16n}}$, i.e., 98% in case of 41 encrypted addresses (in the previous relation, for n IDs, 2^{16n} is the total number of possible addresses and $2^{16}!/(2^{16}-n)!$ is the number of unique addresses). We also provide concrete measurements that confirm these expectations in the experimental section. Storing the tree will require $\Lambda \times m_{node}$ where m_{node} is the memory space required by one node from the tree. The CPU load can be easily expressed as $\mathbb{T}_{\mathsf{comp}}/\Delta$ since one such encrypted address tree has to be computed for each time interval Δ.

As a more concrete representation, in Figure 12 we show how the address map is exported to a circular list for each ID having $\Delta = 1s$ and coverage $\rho = 100\%$. The first ID has a cycle of 100ms and therefore $\lambda_1 = 10$ while the second and third occur at 50ms and 20ms which lead to $\lambda_2 = 20$ and $\lambda_3 = 50$. During each transmission the ID is shifted based on the circular list. The ordered binary tree should be re-computed with new freshness parameters to prevent an adversary for learning the re-mapped IDs. We discuss more on this along with the experimental results but in principle a new address table can be re-generated each second or even faster than that.

To get a more accurate image on these tradeoffs, we now illustrate them by a practical example for the CAN network that is subject to our analysis. In the heavy-duty vehicle there are 33 cyclic IDs originating from three controllers: 2 from the TCM, 10 from the ECM and 21 from the BCM. The rest of the IDs up to 41 are non-cyclic, a case in which we use a single encryption mask. This leads to about 510 encrypted IDs each second in case of a coverage of $\rho = 100\%$. Figure 13 depicts the CPU load in percent for a lifetime $\Delta = 1s$ of the key tree, having a coverage ρ from 10% to 100% (for 10% coverage each ID will be used consecutively 10 times, while for 100% coverage all IDs are unique during one second). The time require for one encryption was selected between $24.8\mu s$ and $2.8ms$ which are the minimum and maximum for one AES computation on our high-end and low-end boards. For the high-end platform the CPU load is less than 0.15% even at 100% coverage. For the low-end platform the CPU load would be 17.92% which is still affordable. These estimations are confirmed by experiments in the next section.

V. Experimental evaluation

In this section we provide experimental data on the effectiveness of the proposed solution.

A. Computational results

As in-vehicle controllers are commonly employed in real-time applications, which highly depend on computational constraints, we now discuss the performance of the encryption procedure. For this, we measure the runtime that is needed for the generation of the ordered binary tree algorithm in order to prove that the proposed method is computationally suitable for automotive grade controllers.

As a representative for high-end automotive boards, we employ a 32-bit S6J32GEKSN microcontroller from Cypress's Traveo Family which runs on a single core ARM cortex R5 with a top operation speed of 240 MHz, while the data and instruction cache are enabled. The microcontroller is equipped with 128 KB RAM and 2048 KB of Flash. Two other high-end candidates used in our evaluation are part of the Infineon TriCore family. The first, a TC1797 microcontroller, is equipped with a TriCore V1.3.1 core that can run at up to 180 MHz and has access to 156 KByte of RAM and 4 MByte of Flash. TC397, the second TriCore platform, is built around a newer generation core which embeds 6 cores running at a maximum frequency of 300 MHz and is equipped with 16 MB of program flash and 2528 KB of SRAM. As a representative for the low-end automotive sector, we use a 16-bit S12XF chip running at up 50 MHz which offers 32KB of RAM and 512KB of Flash.

Table II presents the run-times measured for the generation of the encrypted addresses at various lifetimes Δ and coverage ρ, and the cost of AES (128 bit block and key) and SPECK (64 bit block and 96 bit key) symmetric encryptions. The run-time for the execution of the AES algorithm is $24.8\mu s$-$77.6\mu s$ for the high-end boards, i.e., S6J32GEKSN, TC1797 and TC397, and less than 3ms for the low-end board, i.e., the S12, while SPECK is 5-15 times faster. On the S12 however, SPECK costs the same as the AES likely due to 16 bit architecture of the processor as the code that we use was dedicated for 32-bit platforms and 16-bit conversions automatically handled at the compiler level. The run-time for generating the encrypted addresses varies from $2.16ms$ (high-end controllers) to $246ms$ (low-end controllers) even in case of a full $\rho = 100\%$ coverage. In case of S12XF512 ($\Delta = 10s, \rho = 25\%$), the algorithm does not fit into a single

RAM page and requires around 50% of the RAM which is too demanding for a single task. In overall, key generation requires less than 1% of CPU time/second for high-end cores, and less than 20% CPU time/second for the low-end ones, which is affordable. For security reasons, the address tree should be updated periodically, e.g. every 1–10s, and to avoid de-synchronizations fresh addresses will be generated in advance and newer addresses will replace older ones when they have been consumed. This would require twice the storage space for one address table as depicted in Table II but that is still affordable as it ranges from 0.86–7.47KB.

B. Active defense mechanism

Our experimental setup is shown in Figure 14 and consists of a S6J32GEKSN board responsible for CAN traffic acquisition through the input capture unit (ICU) pin together with a S12 board with adversary capabilities, a power supply and a PicoScope for monitoring the CAN frames. In order to actively detect and eliminate intrusive frames, we embed an application which consists of two components: the CAN frames acquisition/verification procedure and the CAN frame destruction procedure triggered in case of a malicious frame. These components are described next.

CAN frames acquisition At the hardware level, the Rx output from the CAN transceiver is routed to two different inputs of the µC: the actual CAN Rx pin and an ICU pin. Our proposed hardware solution is depicted in Figure 15. The software algorithm is presented in Figure 16. Generally, an ICU, is used to measure the duration of an input signal arriving to its corresponding PIN of the microcontroller. Depending on the configuration the ICU can measure either period or level duration by using a free running timer which is restarted through a flag *TReset* (line 4 from algorithm) for each trigger event (level change). Consequently, we use the ICU to measure any level duration for the received CAN frame, meaning that it detects a number of bits (n) that are either recessive or dominant by dividing the obtained free run timer (FRT) value (in ticks) with the value corresponding to one CAN bit, calculated based on the CAN baud rate, i.e. 4 µs/bit for 250 Kbps equivalent with 240 ticks at 60 MHz peripheral clock denoted as Tbit (line 2-3). Before calculating the number of bits we add half of the value for one bit (120 - line 3) in order to ensure that the full value for one bit is not lost due to the integer division. We also account for eliminating the CAN stuffing bits from the acquisition buffers (line 3, lines 6-9). If the value computed in line 3 is larger than the value corresponding for 5 bits of the same polarity then this value is ignored as it corresponds to an inter-frame space. In this case, we perform the initializations required for a new SOF bit and prepare the indexes in the acquisition buffer to be in line for the next frame (lines 20-22). Note that the variables *stuff* and *polarity* are reset (being global variables, they were initialized to 0 by the RAM initialization routine of the microcontroller). For each frame we reserved 4*32 bits elements (line 22) from the acquisition buffer. The acquisition buffer is accessed through a pointer *p and is filled bit by bit (lines 12-14) corresponding to the calculated number of bits of same polarity

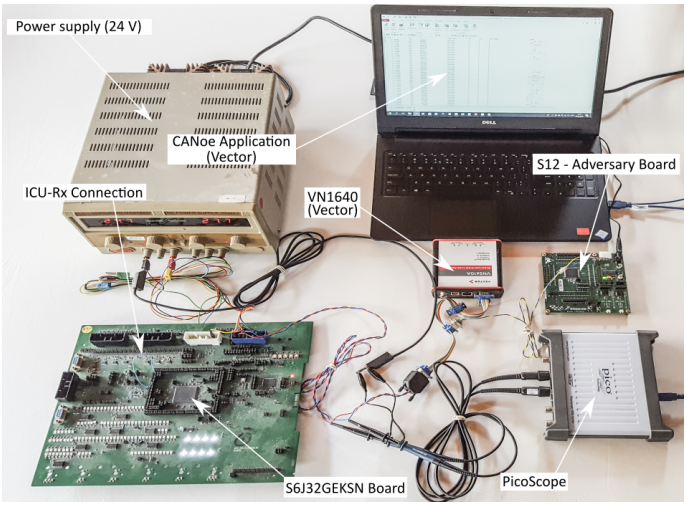


Fig. 14. Experimental setup with the VN1640, S6J32GEKSN and S12 microcontrollers, Power Supply and the PicoScope for CAN traffic monitoring

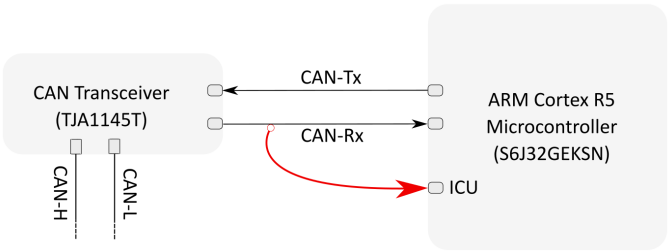


Fig. 15. Architecture of the proposed hardware solution for frame acquisition

(line 3) and the polarity is inverted for the up-coming bits (line 19). The global variable *msg* is used as a message counter inside the buffer and *step* represents the bit position inside the same buffer - these values are initialized when the program starts and will be reset when the buffer is full, e.g., we use a buffer size of 128 frames in our implementation.

CAN frame destruction Once the entire data field is stored in the acquisition buffer, i.e., 103 bits which consist in the SOF(1), arbitration(32), control(6) and data field(64) (line 15 from the algorithm), the ICU detection is stopped in order to process the buffer without any other interruptions from the ICU which may come due to the level changes caused by the reception of the remaining CAN frame fields, e.g. CRC, ACK etc. Firstly, we check the correctness of the ID from the received CAN message, by comparing the received ID with our predefined list of IDs which is stored in memory. This predefined list of IDs is stored in a vector and obtained by AES symmetric encryption as previously discussed. We test equality with the received ID by using an index that corresponds to the column of the matrix. If the ID matches then we increment the index and later reset it when it reaches the maximum value, i.e., the number of precomputed encrypted addresses. In case that the equality check fails, we proceed to destroying the frame to ensure that it is not received by any other ECU. The frame destruction consists in forcing the Tx pin to the dominant state for a period longer than 5 CAN bits (inducing

TABLE II
COMPUTATIONAL RESULTS FOR GENERATING THE CRYPTO-STREAM

	Address Generation Procedure								Encryption	
	$\Delta = 1s, \rho = 25\%$		$\Delta = 1s, \rho = 50\%$		$\Delta = 1s, \rho = 100\%$		$\Delta = 10s, \rho = 25\%$		AES 128/128	SPECK 64/96
Platform	CPU	MEM	CPU	MEM	CPU	MEM	CPU	MEM	CPU	CPU
S6J32GEKSN	$588\mu s$	0.86KB	$1.05ms$	1.49KB	$2.16ms$	2.98KB	$5.84ms$	7.47KB	$24.8\mu s$	$5.32\mu s$
S12XF512	$68ms$		$120ms$		$246ms$		N/A		$2.8ms$	$2.56ms$
Tricore TC1797	$1.76ms$		$2.98ms$		$6.24ms$		$15.20ms$		$77.6\mu s$	$5.30\mu s$
Tricore TC397	$746\mu s$		$1.30ms$		$2.79ms$		$7.37ms$		$28.672\mu s$	$6.97\mu s$

Algorithm 1 CAN frame acquisition

Input: *TValue* (current value of the timer in ticks)
Output: *frame* (content of the ID and datafield)

procedure FRAME ACQUISITION
  $Tbit \leftarrow 240$
  $n \leftarrow (TValue + 120)/Tbit - stuff$
  $TReset \leftarrow 1$
  if $(n \leq 5)$ **then**
    if $(stuff = 1)\&\&(n = 4)$ **then**
      $stuff \leftarrow 1$
    else
      $stuff \leftarrow n/5$
    end if
    for $i = 0, i \leq n$ **do**
      $p \leftarrow \&var[step/32];$
      $*p \leftarrow (*p) \vee ((polarity) \ll (31 - (step\%32)))$
      $step \leftarrow step + 1$
      if $(step \geq (((msg - 1) \ll 7) + 103))$ **then**
        return *frame*
      end if
    end for
    $polarity \leftarrow !polarity$
  else
    $polarity \leftarrow 0, stuff \leftarrow 0$
    $step \leftarrow (msg - 1) * 128$
  end if
end procedure

Fig. 16. Algorithm for CAN frame acquisition

a stuffing error on the bus). The process of destroying CAN frames is depicted in Figure 17 which illustrates the CAN-H and CAN-L lines as monitored by a PicoScope tool set to CAN serial decoding. In the right side of the image we also observe that the decoding of CAN frame is disrupted due to the activation of the error flag. Setting the Tx bit to dominant is done by switching the Tx CAN Pin function to GPIO (General Purpose Input Output) function and output direction. Then we set the PIN value to 0 logic. After the frame destruction we restart the ICU detection in order to be prepared for the next CAN frame.

C. Detecting intrusions based on J1939 specifics

We now discuss how to detect adversarial manipulations based on specific data that can be retrieved from the standardized structure of J1939 frames. For brevity, our analysis is focused on five parameters: i) engine speed (rpm), ii) engine torque (%), iii) fuel consumption (l/h), (iv) vehicle speed (km/h) and (v) engine temperature (°C). This can be of course extended to other values carried by the frames. The first two parameters are carried by the message with ID `0xCF00400` having a cycle time of $20ms$, the third corresponds to ID `0x18FEF200` with cycle time $100ms$, the

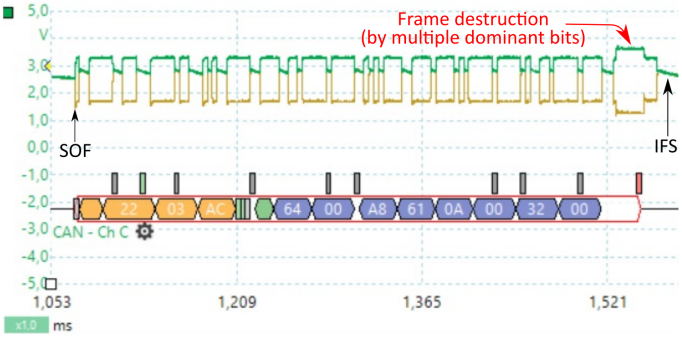


Fig. 17. CAN frame destruction (plot from a digital oscilloscope)

fourth to ID `0x18FEF121` with cycle time $100ms$ while the last one to ID `0x18FEEE00` having a cycle time of $1s$.

The left column of Figure 18 shows the variation of these parameters during 20 minutes of vehicle normal operation. In the right column of Figure 18 we show the variation of differences between consecutive values as a histogram distribution. We note that while the data-range is high, e.g., up to 200 rpm, 20 l/h or 35 km/h at a resolution of up to 16 bits, due to the high acquisition rate the differences between consecutive values are small. This allows us to introduce a simple and effective bounds checking algorithm to spot potential intrusions. For each value $v_i, i = 1..n$ we construct a vector containing the differences recorded during normal runtime, i.e., $\Delta(v_i) = v_i - v_{i-1}, i = 2..n$ and denote as Δ_{min}, Δ_{max} the minimum and maximum values in this array - concrete instantiations are outlined in Table III. Then for each newly received value v_i we check that $v_i \in [v_{i-1} - \Delta_{min}, v_{i-1} + \Delta_{max}]$. The right side of the Figure 18 shows the predicted bounds in green line, i.e., minimum and maximum values, from the currently reported values. By using this bound checking, the intrusion detection algorithm can efficiently detect manipulations. Note that while for engine and vehicle speed both a maximum and minimum range can be checked, for fuel consumption we can check only the maximum value since the inspection of the trace showed that the minimum (zero) is commonly reported regardless of the current value.

D. Accuracy results on detecting intrusions

To assess the effectiveness of the proposed intrusion detection and prevention mechanism, we create a realistic laboratory testbed in which we reconstruct the CAN bus topology inside

TABLE III
OBSERVED VARIATIONS FOR (I) ENGINE SPEED (RPM), (II) FUEL CONSUMPTION (L/H) AND (III) VEHICLE SPEED

Parameter	Unit	Range	Resolution	Δ_{min}	Δ_{max}
Engine speed	rpm	0 to 8031.875	0.125	-35	62
Engine Torque	%	-125 to 125	1	-28	22
Fuel consumption	l/h	0 to 3212.75	0.05	-16	5
Vehicle speed	km/h	0 to 250.996	0.0039	-1	1.6
Engine temperature	°C	-40 to 210	1	-1	1

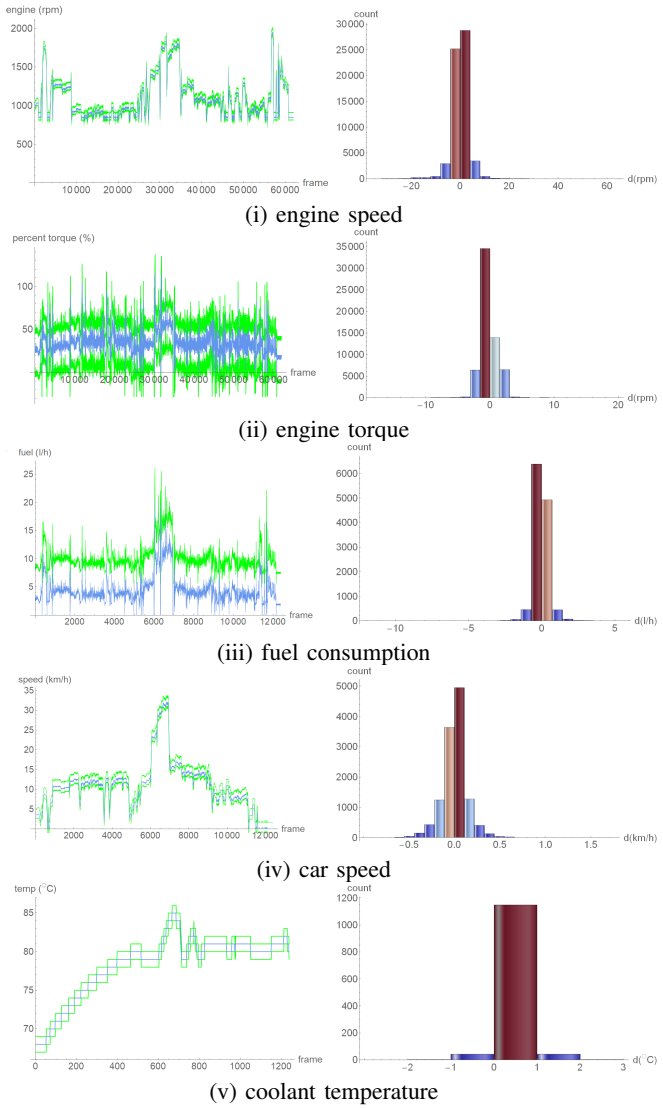


Fig. 18. Predicted min-max bounds of (i) engine speed (rpm), (ii) engine torque (%), (iii) fuel consumption (l/h), (iv) vehicle speed (km/h) and (v) coolant temperature (°C) (left) and histogram distribution of differences between consecutive values (right)

a CANoe simulation. This environment can be used for testing the prevention mechanism without affecting the real vehicle. In this environment we generate traffic that is augmented with adversarial capabilities accounting for frame replays and modifications.

The first protection layer, i.e., the use of encrypted addresses, works against both these types of attacks. Concretely, if 8 bits for each of the sender and destination addresses are used, the probability to guess the correct forthcoming value for an ID is 2^{-16} (assuming one particular ID is targeted). If the attack is not targeted, the highest success probability for the adversary is when using the most frequent start of the IDs, in our case is 0x18FE which occurs for 12 IDs, and guessing the 16 encrypted bits which leads to a cumulative probability of $12 \times 2^{-16} = 0.018\%$ since matching any of the 12 IDs that share the same start is fine for the untargeted adversary. This is roughly the equivalent of a security level of $-\log_2(12/2^{16}) = 12.41$ bits. The second detection layer, i.e., the datafield encryption, offers additional resilience against modifications even when single bits are altered due to the avalanche effect and subsequent range checks on any of the signals will indicate that the frame was altered.

The experimental detection rates are shown in Table IV. For a better image on the equivalent security level, we present it in the table as $\ell = -\log_2(\mathrm{FNR})$. For the first detection layer, i.e., containing *encrypted addressees*, we test the adversary chance to guess any valid ID from the encrypted address matrix by generating addresses at random. The chances of an intrusion to go undetected in this case is as low as 0.015% corresponding to a security level of 11.57 which is close to the theoretical estimation. As for the manipulation of the datafield, the detection accuracy is over 97%, except for the engine torque where the true positive rate (TPR) drops at around 80% with a false negative rate (FNR) of around 20%. Indeed, the variation for this 8 bit parameter is considerably higher than for the others (-28,22) which increases the chances of an adversary to guess a value in this range. Although we have an even higher variation (-35,62) for the engine speed, the adversary chances decrease considerably due to the larger, 16-bit, length of this parameter. The *false positives rate* (FPR) is 0% and the *true negative rate* (TNR) is 100% which means that none of the legitimate frames (containing correct addresses is misclassified).

When both prevention layers are used, the success rate of the adversary becomes so low that it cannot be measured by attack experiments. That is, the chances for an adversary frame to go undetected stays in the range of 0.000159–0.006820% which resulted in our trace for no single intrusion to go undetected. The detection results for the combined layers are synthetically computed in Table V. These values result from multiplying the probability that the adversary ID goes undetected with the probability of the data-field manipulation to go undetected, i.e., multiplying the FNRs for the ID and data-field. The results from Table V decisively prove that an adversary has very small chances for a successful attack. Nonetheless, by using the frame destruction mechanism from the previous section, none of the detected adversary frames will reach the bus.

Overall security level. Our approach has a security level that is compatible with AUTOSAR SecOC [11] demands or may even exceed it. This standard requests for 24-28 bits of authentication data and 0-8 bits of freshness parameter to be carried in inside each frame. The 0-8 bit freshness parameter is already exceeded by the fresh addresses that we store in circular lists. By previous computations, our approach already garners 12 bits of security from the encrypted ID. We have only used the sender and destination addresses in an encrypted form in order to allow message filtering at the CAN driver level

TABLE IV
DETECTION RATES FOR MULTIPLE TARGETED ATTACK PARAMETERS

	Nr. of frames		Detection results				
Attacked param.	Attacks	Genuine	TNR	TPR	FPR	FNR	ℓ (b)
ID	214175	427660	100%	99.99%	0%	0.01%	11.57
Engine speed	15546	61945	100%	97.25%	0%	2.75%	5.18
Engine torque	15446	61945	100%	79.25%	0%	20.75%	2.26
Fuel consumption	3082	12386	100%	99.48%	0%	0.52%	7.58
Vehicle speed	3098	12386	100%	98.61%	0%	1.39%	6.16
Engine temp.	295	1236	100%	98.64%	0%	1.36%	6.2

TABLE V
ESTIMATED DETECTION RATES FOR COMBINED ID & PARAMETER FIELD

Target parameter	TNR	FPR	TPR	FNR	ℓ (b)
Engine speed	100%	0%	99.9991%	0.000895%	16.76
Engine torque	100%	0%	99.9931%	0.006820%	13.83
Fuel consumption	100%	0%	99.9998%	0.000159%	12.61
Vehicle speed	100%	0%	99.9995%	0.000443%	17.78
Engine temp.	100%	0%	99.9995%	0.000443%	17.78

using the remaining 13 bits of the ID. If this is not needed and only the first 3 bits are to be kept for arbitration, all the remaining 26 bits of the ID can be encrypted resulting in a security level of $-\log_2(41/2^{26}) \approx 20$ bits (here we considered that there are 41 available IDs on our network). Even if this is not the case, each field of the payload, as tested in Table IV, contributes with an additional 2-8 bits of security. Worst case, as illustrated by the engine torque field which exhibits the worst performance in detecting injections, this leads to an approximate security level of 2 bits/byte. Note that datafield portions that are constant have an even higher security level of 4 bits/byte since by the avalanche effect 50% of the bits will change. Consequently, at 2 bits/byte we expect the decryption of the datafield to yield an equivalent security level of at least 16 bits for each frame. By adding the 16 bits to the 12 bits from the ID field, we get a security level of 28 bits which perfectly matches the 24-28 bit range of the AUTOSAR SecOC [11]. Thus our solution is effective and corresponds to current security needs. Clearly, our approach is much more secure than regular IDS proposals in [8], [16], [19], [26], etc. which do not use encrypted addresses or the avalanche effect on datafields.

VI. CONCLUSION

Our work provides an efficient intrusion prevention mechanism that is specifically tailored to meet J1939 specifications. Notably, the full allocation of the datafield in J1939 implementations makes it impossible to integrate security elements such as the truncated MACs requested by AUTOSAR specifications [11]. Our proposal circumvents this problem by using encrypted addresses and symmetric encryption on the datafield relying on the avalanche effect of a block cipher that will allow single bit manipulations to be detected by the range-checks of the IDS. Encrypted addresses will hinder an adversary in determining the role of each frame and injecting correct IDs in the network. The proposed defense mechanism is effective and induces small computational costs that can be handled both by low and high-end automotive-grade controllers. The detection procedures keeps a false positive rate of 0% which means that none of the legitimate frames is misclassified and a false negative rate of around 10^{-6} which means that chances for an adversarial frame to be missed are extremely small. Finally our implementation has the ability to destroy malicious frames in real-time with error flags taking advantage of the faster reconstruction with the ICU. Regarding future extensions, very recently, the Society of Automotive Engineers (SAE) has released specifications for CAN-FD in heavy-duty vehicles, i.e., under J1939-17[1], which will open roads for extending regular J1939 frames with security elements. Since the address allocation procedure will likely remain similar, our protocol will still apply to the newer specifications while the generously extended data-field of CAN-FD frames will allow a much easier integration of security elements. The use of this new extension of CAN as well as of cryptographic hardware from embedded boards to speed-up computations is potential future work for us.

Acknowledgement. This work was supported by a grant of the Romanian National Authority for Scientific Research and Innovation, CNCS-UEFISCDI, project number PN-III-P1-1.1.-TE-2016-1317 (2018-2020).

REFERENCES

[1] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham *et al.*, "Experimental security analysis of a modern automobile," in *Security and Privacy (SP), 2010 IEEE Symposium on*. IEEE, 2010, pp. 447–462.

[2] S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, K. Koscher, A. Czeskis, F. Roesner, T. Kohno *et al.*, "Comprehensive Experimental Analyses of Automotive Attack Surfaces," in *USENIX Security Symposium*. San Francisco, 2011.

[3] C. Miller and C. Valasek, "A survey of remote automotive attack surfaces," *Black Hat USA*, 2014.

[4] P. Murvay and B. Groza, "Security Shortcomings and Countermeasures for the SAE J1939 Commercial Vehicle Bus Protocol," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 5, pp. 4325–4339, 2018.

[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)*. Austin, TX: USENIX Association, Aug. 2016. [Online]. Available: https://www.usenix.org/conference/woot16/workshop-program/presentation/burakova

[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*. Cham: Springer International Publishing, 2016, pp. 23–42.

[7] M. Zachos, "Securing J1939 Communications Using Strong Encryption with FIPS 140-2," in *SAE Technical Paper*. SAE International, 03 2017. [Online]. Available: https://doi.org/10.4271/2017-01-0020

[8] S. Mukherjee, J. Walkery, I. Rayz, and J. Daily, "A Precedence Graph-Based Approach to Detect Message Injection Attacks in J1939 Based Networks," in *2017 15th Annual Conference on Privacy, Security and Trust (PST)*, 2017, pp. 67–6709.

[9] H. Shirazi, I. Ray, and C. Anderson, "Using Machine Learning to Detect Anomalies in Embedded Networks in Heavy Vehicles," in *Foundations and Practice of Security*. Cham: Springer International Publishing, 2020, pp. 39–55.

[10] E. Aliwa, O. Rana, C. Perera, and P. Burnap, "Cyberattacks and Countermeasures For In-Vehicle Networks," *arXiv e-prints*, p. arXiv:2004.10781, Apr. 2020.

[11] *Specification of Secure Onboard Communication*, R20-11 ed., AUTOSAR, November 2020, no. 654.

[12] A. Van Herrewege, D. Singelee, and I. Verbauwhede, "CANAuth-a simple, backward compatible broadcast authentication protocol for CAN bus," in *ECRYPT Workshop on Lightweight Cryptography*, vol. 2011, 2011, p. 20.

[1] https://www.sae.org/standards/content/j1939-17_202012/

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

[14] B. Groza, P.-S. Murvay, A. Van Herrewege, and I. Verbauwhede, "LiBrA-CAN: a Lightweight Broadcast Authentication protocol for Controller Area Networks," in *11th International Conference on Cryptology and Network Security, CANS 2012, Springer-Verlag, LNCS*, 2012.

[15] S.-F. Lokman, A. T. Othman, and M.-H. Abu-Bakar, "Intrusion detection system for automotive Controller Area Network (CAN) bus system: a review," *EURASIP Journal on Wireless Communications and Networking*, vol. 2019, no. 1, p. 184, 2019.

[16] M.-J. Kang and J.-W. Kang, "Intrusion detection system using deep neural network for in-vehicle network security," *PloS one*, vol. 11, no. 6, p. e0155781, 2016.

[17] S. Boumiza and R. Braham, "An Anomaly Detector for CAN Bus Networks in Autonomous Cars based on Neural Networks," in *2019 International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*. IEEE, 2019, pp. 1–6.

[18] D. Tanaka, M. Yamada, H. Kashima, T. Kishikawa, T. Haga, and T. Sasaki, "In-Vehicle Network Intrusion Detection and Explanation Using Density Ratio Estimation," in *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*. IEEE, 2019, pp. 2238–2243.

[19] L. Gao, F. Li, X. Xu, and Y. Liu, "Intrusion detection system using SOEKS and deep learning for in-vehicle security," *Cluster Computing*, vol. 22, no. 6, pp. 14 721–14 729, 2019.

[20] G. Loukas, T. Vuong, R. Heartfield, G. Sakellari, Y. Yoon, and D. Gan, "Cloud-based cyber-physical intrusion detection for vehicles using deep learning," *IEEE Access*, vol. 6, pp. 3491–3508, 2017.

[21] C. Wang, Z. Zhao, L. Gong, L. Zhu, Z. Liu, and X. Cheng, "A distributed anomaly detection system for in-vehicle network using HTM," *IEEE Access*, vol. 6, pp. 9091–9098, 2018.

[22] H. Lee, S. H. Jeong, and H. K. Kim, "OTIDS: A novel intrusion detection system for in-vehicle network by using remote frame," in *2017 15th Annual Conference on Privacy, Security and Trust (PST)*. IEEE, 2017, pp. 57–5709.

[23] Y. Yang, L. Wang, Z. Li, P. Shen, X. Guan, and W. Xia, "Anomaly Detection for Controller Area Network in Braking Control System With Dynamic Ensemble Selection," *IEEE Access*, vol. 7, pp. 95 418–95 429, 2019.

[24] W. Choi, K. Joo, H. J. Jo, M. C. Park, and D. H. Lee, "Voltageids: Low-level communication characteristics for automotive intrusion detection system," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 8, pp. 2114–2129, 2018.

[25] J. Ning, J. Wang, J. Liu, and N. Kato, "Attacker Identification and Intrusion Detection for In-Vehicle Networks," *IEEE Communications Letters*, vol. 23, no. 11, pp. 1927–1930, 2019.

[26] B. Groza and P. Murvay, "Efficient Intrusion Detection With Bloom Filtering in Controller Area Networks," *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 4, pp. 1037–1051, 2019.

[27] M. M¨uter and N. Asaj, "Entropy-based anomaly detection for in-vehicle networks," in *2011 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2011, pp. 1110–1115.

[28] M. Marchetti, D. Stabili, A. Guido, and M. Colajanni, "Evaluation of anomaly detection for in-vehicle networks through information-theoretic algorithms," in *2016 IEEE 2nd International Forum on Research and Technologies for Society and Industry Leveraging a better tomorrow (RTSI)*. IEEE, 2016, pp. 1–6.

[29] Q. Wang, Z. Lu, and G. Qu, "An entropy analysis based intrusion detection system for controller area network in vehicles," in *2018 31st IEEE International System-on-Chip Conference (SOCC)*. IEEE, 2018, pp. 90–95.

[30] W. Wu, Y. Huang, R. Kurachi, G. Zeng, G. Xie, R. Li, and K. Li, "Sliding Window Optimized Information Entropy Analysis Method for Intrusion Detection on In-Vehicle Networks," *IEEE Access*, vol. 6, pp. 45 233–45 245, 2018.

[31] R. Islam, R. U. D. Refat, S. M. Yerram, and H. Malik, "Graph-Based Intrusion Detection System for Controller Area Networks," *IEEE Transactions on Intelligent Transportation Systems*, pp. 1–10, 2020.

[32] H. M. Song, H. R. Kim, and H. K. Kim, "Intrusion detection system based on the analysis of time intervals of CAN messages for in-vehicle network," in *2016 international conference on information networking (ICOIN)*. IEEE, 2016, pp. 63–68.

[33] H. Ji, Y. Wang, H. Qin, Y. Wang, and H. Li, "Comparative Performance Evaluation of Intrusion Detection Methods for In-vehicle Networks," *IEEE Access*, vol. PP, pp. 1–1, 06 2018.

[34] K.-T. Cho and K. G. Shin, "Fingerprinting electronic control units for vehicle intrusion detection," in *25th {USENIX} Security Symposium ({USENIX} Security 16)*, 2016, pp. 911–927.

[35] A. Taylor, S. Leblanc, and N. Japkowicz, "Anomaly detection in automobile control network data with long short-term memory networks," in *2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*. IEEE, 2016, pp. 130–139.

[36] A. Taylor, N. Japkowicz, and S. Leblanc, "Frequency-based anomaly detection for the automotive CAN bus," in *2015 World Congress on Industrial Control Systems Security (WCICSS)*, 2015, pp. 45–49.

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

[38] "ISO11898-1. Road vehicles — Controller area network (CAN) —Part 1: Data link layer and physical signalling," International Organization for Standardization, Standard, 2nd edition, Dec. 2015.

[39] "ISO11898-2. Road vehicles — Controller area network (CAN) — Part 2: High-speed medium access unit," International Organization for Standardization, Standard 2nd edition, Dec. 2016.

[40] "J1939-21 – Data Link Layer," SAE International, Standard, Dec. 2006.

[41] "J1939-71 – Vehicle Application Layer," SAE International, Standard, May 2006.

[42] "Digital Annex of Serial Control and Communication," SAE International, Standard, Jan. 2020.

[43] "J1939-81 – Network Management," SAE International, Standard, Mar. 2003.

[44] "J1939-13 – Off-Board Diagnostic Connector," SAE International, Standard, Mar. 2004.

[45] H. M. Song, H. R. Kim, and H. K. Kim, "Intrusion detection system based on the analysis of time intervals of CAN messages for in-vehicle network," in *Information Networking (ICOIN), 2016 International Conference on*. IEEE, 2016, pp. 63–68.

[46] K.-T. Cho and K. G. Shin, "Error handling of in-vehicle networks makes them vulnerable," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2016, pp. 1044–1055.

[47] A. Humayed and B. Luo, "Using ID-hopping to defend against targeted DoS on CAN," in *Proceedings of the 1st International Workshop on Safe Control of Connected and Autonomous Vehicles*. ACM, 2017, pp. 19–26.

[48] W. Wu, R. Kurachi, G. Zeng, Y. Matsubara, H. Takada, R. Li, and K. Li, "IDH-CAN: A Hardware-Based ID Hopping CAN Mechanism With Enhanced Security for Automotive Real-Time Applications," *IEEE Access*, vol. 6, pp. 54 607–54 623, 2018.

[49] S. Woo, D. Moon, T.-Y. Youn, Y. Lee, and Y. Kim, "CAN ID Shuffling Technique (CIST): Moving Target Defense Strategy for Protecting In-Vehicle CAN," *IEEE Access*, vol. 7, pp. 15 521–15 536, 2019.

[50] B. Groza, L. Popa, and P.-S. Murvay, "Highly Efficient Authentication for CAN by Identifier Reallocation with Ordered CMACs," *IEEE Transactions on Vehicular Technology*, 2020.

[51] R. Beaulieu, D. Shors, J. Smith, S. Treatman-Clark, B. Weeks, and L. Wingers, "SIMON and SPECK: Block Ciphers for the Internet of Things," *IACR Cryptol. ePrint Arch.*, vol. 2015, p. 585, 2015.

[52] S. Jain and J. Guajardo, "Physical layer group key agreement for automotive controller area networks," in *International Conference on Cryptographic Hardware and Embedded Systems*. Springer, 2016, pp. 85–105.

[53] D. Püllen, N. A. Anagnostopoulos, T. Arul, and S. Katzenbeisser, "Using Implicit Certification to Efficiently Establish Authenticated Group Keys for In-Vehicle Networks," in *2019 IEEE Vehicular Networking Conference (VNC)*, 2019, pp. 1–8.

[54] N. K. Giri, A. Munir, and J. Kong, "An Integrated Safe and Secure Approach for Authentication and Secret Key Establishment in Automotive Cyber-Physical Systems," in *Intelligent Computing*, 2020, pp. 545–559.

[55] T. Tatara, H. Ogura, Y. Kodera, T. Kusaka, and Y. Nogami, "Updating A Secret Key for MAC Implemented on CAN Using Broadcast Encryption Scheme," in *2019 34th International Technical Conference on Circuits/Systems, Computers and Communications*, 2019, pp. 1–4.

Camil Jichici is a PhD student at Politehnica University of Timisoara (UPT) since 2018 and works as a young researcher in the PRESENCE project. He received the Dipl.Ing. degree in 2016 and MsC. degree in 2018, both from UPT. His research interests are on the security of in-vehicle components and networks. He is also working as a software integrator in the automotive industry for Continental Corporation in Timisoara since 2014.

Bogdan Groza is 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. Besides regular participation in national and international research projects in information security, he lead the CSEAMAN project (2015-2017) and currently leads the PRESENCE project (2018-2019), two research programs dedicated to automotive security funded by the Romanian National Authority for Scientific Research and Innovation.

Radu Ragobete is a Senior Software Engineer working for the automotive industry. He received the Dipl.Ing. and MsC. degree from Politehnica University of Timisoara (UPT) in 2006 and 2008 respectively. An employee of Continental Automotive in Timisoara (Romania) since 2006, his main responsibilities are in developing low-level drivers and implementation of platform functional safety requirements. His main interests include the microcontroller startup process and automotive operating systems.

Pal-Stefan Murvay is a Lecturer 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 10-year background as a software developer in the automotive industry. He worked as a postdoctoral researcher in the CSEAMAN project and is currently a senior researcher in the PRESENCE project. He also leads the SEVEN project related to automotive and industrial systems security. His current research interests are in the area of automotive security.

Tudor Andreica is a Ph.D. student at Politehnica University of Timisoara. He graduated his B.Sc and M.Sc studies in 2016 and 2018 respectively, from Polithenica University of Timisoara. Since 2015 he is working as software engineer at HELLA Romania focusing on the security of various in-vehicle systems. He was a research student in the CSEAMAN project and currently joined the PRESENCE project as a PhD student. His research interests are in the field of automotive cybersecurity.

APPENDIX A - IDENTIFIED FRAMES ACCORDING TO J1939 SPECIFICATIONS

This Appendix summarizes the IDs found in the collected traffic from our test vehicle and their role according to the J1939 specification. Further, we detail the data-field for the IDs that were used in our intrusion detection system. Some brief comments on these follow.

In Table VI we observe messages which are specific for address claiming procedure (rows 1-3) and their specific PGN request (rows 4-5 and 28). Many other IDs follow carrying information related to engine, transmission, exhaust and air supply systems, etc. Messages from two distinct PGNs, i.e., the Engine Configuration 1 and Vehicle Identification Number (VIN) depicted in rows 32-33 from Table VI, are sent by the ECM as multi-frame messages - also a J1939 specific. Besides the ECM, the VIN is also transmitted by the BCM, i.e., rows 34-35. The specific on-board diagnostic subsystem can be identified in the collected traffic by the two specific diagnostic J1939 frames (on rows 7-8) which encloses the active diagnostic trouble codes that are sent by BCM and ECM. Table VI does not include the 16 IDs which are OEM specific since their role is not specified in the standard.

Table VII outlines the significance for each parameter from the data field in case of the 4 identifiers that are used in our IDS. The data ranges for these parameters and the transmission rates employed are presented according to [42].

TABLE VI
IDENTIFIED J1939 FRAMES FROM THE COLLECTED CAN TRAFFIC IN ACCORDANCE TO [42]

No.	Pr.	ID	PF	PS	PDU1	PDU2	DA	GE	SA	TP PGN	PG description
1.	6	0x18EEFF03	238	255	✓	–	GLB	–	TCM	–	Address Claimed
2.	6	0x18EEFF00	238	255	✓	–	GLB	–	ECM	–	Address Claimed
3.	6	0x18EEFF21	238	255	✓	–	GLB	–	BCM	–	Address Claimed
4.	6	0x18EAFF00	234	255	✓	–	GLB	–	ECM	–	PGN Request
5.	6	0x18EAFF03	234	255	✓	–	GLB	–	TCM	–	PGN Request
6.	6	0x18FE0F21	254	15	–	✓	–	15	BCM	–	Language Command
7.	6	0x18FECA03	254	202	–	✓	–	202	TCM	–	Active Diagnostic Trouble Codes (Diagnostic message 1)
8.	6	0x18FECA21	254	202	–	✓	–	202	BCM	–	Active Diagnostic Trouble Codes (Diagnostic message 1)
9.	6	0x18FEF200	254	242	–	✓	–	242	ECM	–	Fuel Economy (Liquid)
10.	3	0xCF00400	240	4	–	✓	–	4	ECM	–	Electronic Engine Controller 1
11.	6	0x18FEF121	254	241	–	✓	–	241	BCM	–	Cruise Control/Vehicle Speed
12.	7	0x1CFEC303	254	195	–	✓	–	195	TCM	–	Electronic Transmission Controller 5
13.	3	0xCF00300	240	3	–	✓	–	3	ECM	–	Electronic Engine Controller 2
14.	6	0x18F00503	240	5	–	✓	–	5	TCM	–	Electronic Transmission Controller 2
15.	6	0x18FEDF00	254	223	–	✓	–	223	ECM	–	Electronic Engine Controller 3
16.	3	0xCFE4521	254	69	–	✓	–	69	BCM	–	Primary or Rear Hitch Status
17.	6	0x18FEF021	254	240	–	✓	–	240	BCM	–	Power Takeoff Information 1
18.	6	0x18FEF021	254	239	–	✓	–	239	ECM	–	Engine Fluid Level/Pressure 1
19.	3	0xCFE4421	254	68	–	✓	–	68	BCM	–	Secondary or Front Power Take off Output Shaft
20.	3	0xCFE4321	254	67	–	✓	–	67	BCM	–	Primary or Rear Power Take off Output Shaft
21.	6	0x18FEF600	254	246	–	✓	–	246	ECM	–	Intake/Exhaust Conditions 1
22.	6	0x18FEAE21	254	174	–	✓	–	174	BCM	–	Air Supply Pressure
23.	7	0x1CFDDF21	253	223	–	✓	–	223	BCM	–	Front Wheel Drive Status
24.	6	0x18FEFC21	254	252	–	✓	–	252	BCM	–	Dash Display 1
25.	6	0x18FEF721	254	247	–	✓	–	247	BCM	–	Vehicle Electrical Power 1
26.	6	0x18F00621	240	6	–	✓	–	6	BCM	–	Electronic Axle Controller 1
27.	3	0xCFDCC21	253	204	–	✓	–	204	BCM	–	Operators External Light Controls Message
28.	6	0x18EAFF21	234	255	✓	–	GLB	–	BCM	–	PGN Request
29.	6	0x18FEE500	254	229	–	✓	–	229	ECM	–	Engine Hours, Revolutions
30.	6	0x18FEEE00	254	238	–	✓	–	238	ECM	–	Engine Temperature 1
31.	6	0x18FEF700	254	247	–	✓	–	247	ECM	–	Vehicle Electrical Power 1
32.	7	0x1CECFF00	236	255	✓	–	GLB	–	ECM	CM.BAM	Engine Configuration 1, Vehicle Identification Number
33.	7	0x1CEBFF00	235	255	✓	–	GLB	–	ECM	TP.DT	Engine Configuration 1, Vehicle Identification Number
34.	7	0x1CECFF21	236	255	✓	–	GLB	–	BCM	CM.BAM	Vehicle Identification Number
35.	7	0x1CEBFF21	235	255	✓	–	GLB	–	BCM	TP.DT	Vehicle Identification Number

TABLE VII
FIELD DETAILS FOR SOME OF THE COLLECTED IDS IN ACCORDANCE TO [42]

No.	PGN/ID	Cycle	Byte pos.	Parameters description	Data Range
1.	65266	100 ms	B1: 2 bytes	Engine Fuel Rate	0 to 3,212.75 l/h
	0x18FEF200		B3: 2 bytes	Engine Instantaneous Fuel Economy	0 to 125.498046875 km/L
			B5: 2 bytes	Engine Average Fuel Economy	0 to 125.498046875 km/L
			B7: 1 byte	Engine Throttle Valve 1 Position 1	0 to 100%
			B8: 1 byte	Engine Throttle Valve 2 Position	0 to 100%
2.	61444	10-50 ms	B1: 4/4 bits	Engine Torque Mode / Actual Engine - Percent Torque (Fractional)	0 to 15
	0xCF00400		B2: 1 byte	Driver's Demand Engine - Percent Torque	-125 to 125 %
			B3: 1 byte	Actual Engine - Percent Torque	-125 to 125 %
			B4: 2 bytes	Engine Speed	0 to 8,031.875 rpm
			B6: 1 bytes	Source Address of Controlling Device for Eng. Control	0 to 255
			B7: 4 bits	Engine Starter Mode	0 to 15
			B8: 1 byte	Engine Demand – Percent Torque	-125 to 125 %
3.	65265	100 ms	B1: 2/2/2/2 bits	Switch: Speed Axle / Park Brake / Cruise Control / Park Brake Release Inhibit	0 to 3
	0x18FEF121		B2: 2 bytes	Wheel-Based Vehicle Speed	0 to 250.996 km/h
			B4: 2/2/2/2 bits	Cruise Control (CC) Active / CC Enable Switch / Brake Switch / Clutch Switch	0 to 3
			B5: 2/2/2/2 bits	CC Set Switch / CC Coast (Dec.) Switch / CC Resume Switch / CC Acc. Switch	0 to 3
			B6: 1 byte	Cruise Control Set Speed	0 to 250 km/h
			B7: 5/3 bits	PTO Governor State / Cruise Control States	0-31/0-7
			B8: 2/2/2/2 bits	Engine: Idle Increment / Idle Decrement / Diagnostic Test / Shutdown Override	0 to 3
4.	65262	1s	B1: 1 byte	Engine Coolant Temperature	-40 to 210 °C
	0x18FEEE00		B2: 1 byte	Engine Fuel 1 Temperature 1	-40 to 210 °C
			B3: 2 bytes	Engine Oil Temperature 1	-273 to 1734.96875 °C
			B5: 2 bytes	Engine Turbocharger 1 Oil Temperature	-273 to 1734.96875 °C
			B7: 1 byte	Engine Intercooler Temperature	-40 to 210 °C
			B8: 1 byte	Engine Charge Air Cooler Thermostat Opening	0 to 100 %