A Simple Intrusion Detection Method for Controller Area Network

Aymen Boudguiga*, Witold Klaudel†, Antoine Boulanger‡ and Pascal Chiron‡
*IRT SystemX, 8 avenue de la Vauve 91120–Palaiseau, *aymen.boudguiga@irt-systemx.fr*
†Renault, 1 avenue du Golf 78288–Guyancourt, *witold.klaudel@renault.com*
‡PSA Peugeot Citroën, Route de Gisy 78140–Velizy-Villacoublay, *firstName.lastName@mpsa.com*

***Abstract*—The Controller Area Network (CAN) is established as the main communication channel inside vehicles. CAN relies on frame broadcast to share data between different microcontrollers managing critical or comfort functions such as cruise control or air conditioning. CAN is distinguished by its simplicity, its real-time application compatibility and its low deployment cost. However, CAN major drawback is its lack of security support. That is, CAN fails to provide protections against attacks such as intrusion, denial of service or impersonation. We propose in this work a simple intrusion detection method for CAN. Our main idea is to make each microcontroller monitor the CAN bus to detect malicious frames.**

***Index Terms*—CAN, ECU, Intrusion Detection, HSM, MAC**

I. Introduction

Advanced Driver Assistance Systems (ADAS) are the most emerging technologies in automotive electronics. Initially, ADAS were designed to alert drivers about prospective hazards and accidents. Nowadays, ADAS provide emergency braking, auto-parking or automatic lane keeping. Future ADAS are evolving toward autonomous driving. They will benefit from wireless car-to-car and car-to-infrastructure connectivities provided by Cooperative Intelligent Transport System (C-ITS). However, the combination of wireless connectivity and automatic driving capabilities creates new threats for vehicle and driver safety, and raises new challenges for securing cars embedded networks.

Figure 1 gives an example of a vehicle embedded architecture adapted to the C-ITS environment. C-ITS relies on two types of access units: Road Side Units (RSUs) installed on roads and On-Board Units (OBUs) embedded in cars. OBUs encompass all vehicles internal networks together with their connected Electronic Control Units (ECUs). The vehicle internal network is formed by various communication buses such as Ethernet and Controller Area Network (CAN) bus [1]. Figure 1 depicts two types of CAN connecting a number of Electronic Control Units (ECUs). In the presented architecture, the Vehicle Control Unit (VCU) is the centric equipment managing the car safety and security. For example, VCU is in charge of car initialization after the driver recognition based on an intelligent key/card authentication. The *On Board Diagnostics* (OBD) plug is another component that is connected to the CAN bus. This plug provides access to diagnostics features necessary for the car control or troubleshooting.

When CAN was initially specified, cars were considered as closed and isolated systems and security concerns were somehow ignored. CAN was designed to be simple, efficient

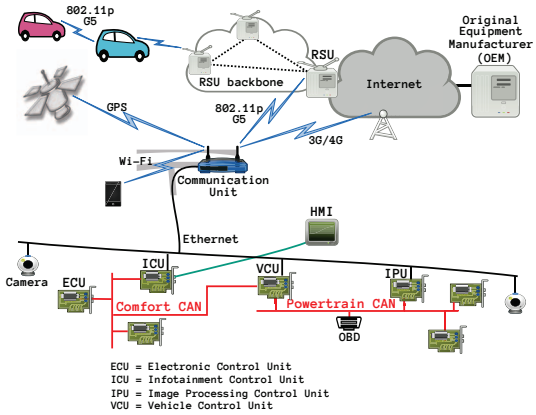

Fig. 1. An example of C-ITS architecture

and ensured low deployment costs. However, few years later, hackers started using the OBD plug as an entry point to the car network. *White hats* aimed at tuning their own cars performances. For example, they modified the engine ECU configuration to increase engine power or reduce its fuel consumption. Meanwhile, *black hats* aimed for car theft, and targeted drivers and passengers safety by attacking critical ECUs.

CAN attacks have been already investigated in literature. In 2008, Hoppe et al. [2] performed a successful attack against the ECU that lifts windows. Meanwhile, Nilsson and Larson [3] presented a virus to eavesdrop messages over the CAN bus. In 2010, Koscher et al. [4] presented an outstanding analysis of CAN security. They executed attacks such as CAN sniffing, CAN spoofing and malware installing on safety related ECUs. They noticed that no security mechanisms were applied during software updates. In 2011, Checkoway et al. [5] extended the analysis to external attack surface on a modern vehicle. Their results were again alarming. Indeed, they compromised car radio using a tampered CD. They even controlled the car telematics via a call to car's integrated cellular phone. Then, they were able to unlock car doors and inhibit anti-theft measures. Recently, in Black Hat 2015, Miller and Valasek demonstrated how they hacked a Jeep Cherokee ECUs via a remote exploitation. They controlled the steering, braking, acceleration and display systems [6].

We propose in this work an intrusion detection method for CAN. Our method detects an attacker which is impersonating as a legitimate ECU. Impersonation is done either

978-1-4799-6664-6/16/$31.00 ©2016 IEEE

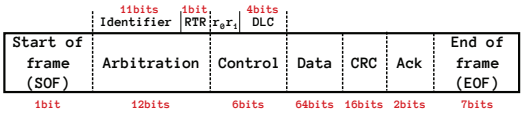

Fig. 2. CAN Data frame

by forging or by replaying valid CAN frames. CAN Data or Remote frame contains a unique identifier that is associated to a unique ECU. Our intrusion detection mechanism makes each legitimate ECU register periodically with other ECUs. Then, the registered ECU monitors the CAN bus to check that no Data frames are being sent on its behalf. That is, the authenticated ECU checks if received Data frames do not contain one of its own frame identifiers. In case of attack, the ECU erases the intruder frame by sending an Error frame. Then, it notifies other ECUs about the detected intrusion.

The remainder of the paper is organized as follows. Section II reviews CAN main features. Section III describes the state of the art regarding CAN authentication, key distribution and intrusion detection. Section IV presents our chosen attacker model and the list of considered attacks on CAN bus. Section V depicts in details our proposed intrusion detection system. Finally, Section VI concludes the paper.

II. Controller Area Network

The Controller Area Network (CAN) connects Electronic Control Units (ECUs) via a broadcast bus. Each ECU is in charge of controlling actuators or sensors. ECUs communicate by exchanging CAN frames. These frames can be of the following types: Data, Remote, Error or Overload. Data frames exchange data between ECUs. Remote frames request the transmission of a specific Data frame. Remote frames have the same format as Data frames (Figure 2). However, they have a *recessive* Remote Transmission Request (RTR) field and an empty payload. Error frames warn about the existence of errors on the CAN bus. Finally, Overload frames serve to delay next frames transmission.

CAN frames do not contain information about their sender i.e. source or receiver i.e. destination. They do not rely on interface addressing. In practice, each ECU manages a limited set of *unique* and *distinct* frame *identifiers* (Figure 2). A frame identifier ($\mathsf{ID_f}$) distinguishes a unique payload information that interests a set of receivers. We call $\mathsf{ID_f}$ *producer* the node broadcasting Data frames having a given $\mathsf{ID_f}$. Meanwhile, we refer by $\mathsf{ID_f}$ *consumers* to the nodes reading Data frames containing a given $\mathsf{ID_f}$. The consumers of frames produced by an $\mathsf{ECU_i}$ form a unique set of consumers $\mathsf{SetConsumers_i}$. Consumers can belong simultaneously to different sets of consumers associated to different producers.

We denote by $\mathsf{SetPID_i} = \{\mathsf{P_{i1}}, \ldots, \mathsf{P_{in}}\}$ the set of Data frames identifiers that are produced by $\mathsf{ECU_i}$. Meanwhile, we define as $\mathsf{SetCID_i} = \{\mathsf{C_{i1}}, \ldots, \mathsf{C_{in}}\}$ the set of Data frames that are consumed by $\mathsf{ECU_i}$. That is, $\mathsf{C_{ik}}$ corresponds to a frame identified with $\mathsf{ID_f} = \mathsf{P_{jl}}$ and produced by a distinct $\mathsf{ECU_{j,j\neq i}}$, i.e. $\mathsf{C_{ik}} = \mathsf{P_{jl}}$ and $\mathsf{ECU_i} \in \mathsf{SetConsumers_j}$.

Figure 3 is an example of frame transmission on the CAN bus. $\mathsf{ECU_1}$ sends a CAN Data frame ($\mathsf{frame_{12}}$) identified by $\mathsf{ID_f} = \mathsf{P_{12}}$. $\mathsf{frame_{12}}$ is only recovered by $\mathsf{ECU_3}$ as it

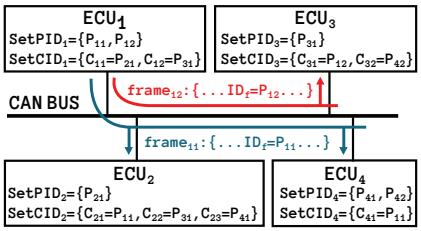

Fig. 3. CAN Data frames transmission

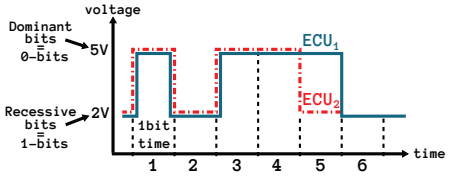

Fig. 4. CAN bitwise arbitration

has $\mathsf{C_{31}} = \mathsf{P_{12}}$ in its $\mathsf{SetCID_3}$. However, When $\mathsf{ECU_1}$ sends $\mathsf{frame_{11}}$, it is recovered by $\mathsf{ECU_2}$ and $\mathsf{ECU_4}$ because they have $\mathsf{P_{11}}$ in their $\mathsf{SetCID_2}$ and $\mathsf{SetCID_4}$, respectively. Note that $\mathsf{ECU_2}$ or $\mathsf{ECU_4}$ can request the retransmission of $\mathsf{frame_{11}}$ by sending a Remote frame. The latter is a Data frame with $\mathsf{ID_f} = \mathsf{P_{11}}$ and a *recessive* RTR bit (Figure 2). At the reception of the Remote frame with $\mathsf{ID_f} = \mathsf{P_{11}}$, $\mathsf{ECU_1}$ retransmits $\mathsf{frame_{11}}$.

CAN bits respect a Non-Return to Zero (NRZ) coding where 0-bits and 1-bits are encoded with different non-null voltage. The value of a transmitted bit is sampled at the end of a *nominal bit time* i.e. a bit time slot. In practice, CAN 0-bits are referenced as *dominant* bits and 1-bits as *recessive* bits. In addition, *bit stuffing* is applied to the start of frame, arbitration, control, data and CRC fields of a CAN frame (Figure 2). Bit stuffing consists on inserting a complementary bit after five consecutive bits of same value. For example, we insert a 1-bit after the transmission of 5 0-bits. The only exception to bit stuffing are the Error and Overload frames which are identified by flags of 6 consecutive bits of same polarity.

CAN relies on CSMA/CA with a *bitwise arbitration* as bus access method. The bitwise arbitration concerns the value of frame identifiers. When two ECUs start the delivery of two different frames at the same time, the ECU sending the frame with the greatest $\mathsf{ID_f}$ value stops its transmission at the reception of a dominant bit while it is transmitting a recessive bit. In Figure 4 example, $\mathsf{ECU_2}$ stops the transmission of its frame after detecting a dominant bit transmission at time slot 5 while transmitting a recessive bit. Consequently, $\mathsf{ECU_1}$ keeps the channel for itself and frame collision is avoided.

III. CAN Network Security

The CAN bus is well-suited for *real time* applications where data transmission delay is a big concern. The real time constraint and the small size of Data frame payload (64-bits) make the introduction of security mechanisms in CAN specification a tough problem. The integration of cryptographic algorithms to provide message integrity and confidentiality

is complex. On the one hand, it is impossible to use *asymmetric* cryptography to provide ECUs authentication and frames integrity or confidentiality. Indeed, the output length of asymmetric signature and encryption algorithms is longer than one CAN payload i.e. 64-bits. For example, RSA signature length is at least 1024-bits while ECDSA signature length is at least 160-bits. Consequently, not only the number of CAN frames needed to send a signature increases but also data processing and transmission time raises due to signature verification time. On the other hand, the use of *symmetric* cryptography for ECUs authentication in a broadcast context has to thwart the insider attack. The use of a shared key for an authentication within a group of nodes is deprecated because any node can impersonate as the key owner and compute valid Message Authentication Codes (MACs). Sharing a symmetric key between more than two nodes does not provide non-repudiation property.

In 2008, Oguma et al. [7] proposed a polynomial key distribution scheme to share pairwise keys between all ECUs. They relied on a master ECU to authenticate other ECUs and to provide them with a session secret. The secret is used later during ECUs pairwise authentication. Oguma et al. solution provides anti-replay attacks thanks to the use of counters. It also avoids malware installation by including a proof of authenticity for each ECU ROM code. In 2009, Szilagyi and Koopman [8] proposed to pre-share pairwise keys between ECUs. Then, a frame producer concatenates in the frame payload as many MACs as receivers. For example, if the frame payload contains 32 bits of data and the number of the frame consumers is equal to 4, then the 32 remaining bits of the payload will contain 4 concatenated MACs, each 8 bits long. Each MAC is computed with the pairwise key that is shared with the corresponding consumer. Schweppe et al. [9] used symmetric keys and MACs to provide CAN frames authenticity. They used truncated MACs of 32 bits long. They were the pioneers to define a Hardware Security Module (HSM) for keys secure storage and tagging. They associated a tag to every key to indicate either it was needed for signing or for verifying a signature. As such, they thwarted the insider attack.

In 2012, Wei and Sangionvanni-Vincentelli [10] relied on pairwise keys and MACs to provide CAN frames authenticity. They sent a separate signed CAN frame to each consumer. In addition, they included a counter in the frame payload to prove its freshness. To avoid counter rollover problem, they sent only the Least Significant Bits (LSB) of the counter in the CAN frame. Meanwhile, they kept secret the counter Most Significant Bits (MSB). A frame consumer accepts frames with a counter value superior to its local one. Groza et al. [11] proposed a master/slave model to distribute keys between ECUs. Slaves must register to the master to get their symmetric keys. Once keys are received, frame producer concatenates MACs, destined to the frame consumers, in the same frame payload. In 2013, Groza and Murvay [12] used key chains and time synchronization to implement a broadcast authentication protocol. Their solution was inspired from the TESLA protocol [13].

All the aforementioned solutions rely on pairwise keys and MACs to mutually authenticate ECUs. Meanwhile, other work investigated Intrusion Detection Systems (IDS) for the CAN bus. For example, in 2010, Hoppe et al. [14] described three intrusion detection patterns for CAN networks. The first pattern evaluates the frequency of messages. The second pattern studies the possible misuse of frame identifiers. The last pattern learns the communication characteristics of each ECU. Examples of such characteristics are fingerprints of transmitted signals. This last approach requires a training phase to learn all ECUs communication characteristics. In 2011, Kleberger et al. [15] defined two intrusion detection methods: specification-based and anomaly-based. The specification-based detection evaluates deviations from CAN specification and ECUs expected behavior. It was investigated in depth by Larson et al. [16]. The latter implemented in each ECU a dedicated entity called the *detector* to spot any deviation from CAN specification. Meanwhile, the anomaly-based detection estimates deviations regarding the number of transmitted messages. It refers to Hoppe et al. [14] analysis of messages frequency. Hoppe et al. [17] defined also a method for an adaptive dynamic reaction to incidents in in-vehicle network via visual, acoustic or haptic notifications.

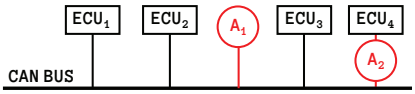

Fig. 5. Attacker position in CAN bus

IV. CAN Attacker Model

In this work, we consider a Dolev and Yao attacker model [18]. That is, the attacker is able to Read, Drop and Send valid CAN frames. A Read action refers to receiving or intercepting frames. Meanwhile, a Send action refers to forging and replaying frames. A Drop action refers to frame filtering. The attacker can connect to the CAN bus at two different points as presented in Figure 5. Attackers A_1 and A_2 target all the ECUs. In addition, the attacker A_2 can isolate ECU_4. That is, A_2 is not only able to read and send frames on behalf of ECU_4 but she can also filter and drop frames going to or coming from ECU_4. In a C-ITS context, attacks are executed by directly accessing the vehicle embedded network or by controlling intermediary components that can be remotely activated.

In this work, we are interested in the following attacks on CAN bus:

- **Denial of Services attack (DoS):** we distinguish two types of DoS: BUS DoS and ECU DoS. BUS DoS is a trivial attack aiming at preventing ECUs from accessing the CAN bus. The attacker has just to connect to the bus and send a sequence of dominant bits, Error frames or fake Data frames with priority identifier to monopolize the channel. Consequently, she prevents legitimate ECUs from sending frames. For autonomous vehicles in a C-ITS context, a BUS DoS attack immobilizes the vehicle. In risk analysis, we talk about an attack having an *Operational* impact [19]. That is, this attack affects the returned quality of service of the vehicle and so the manufacturer corporate image. In practice, BUS DoS cannot be thwarted.

ECU DoS attack targets a unique ECU. For example, the attacker installs a malware in the Brake ECU (BECU) or floods it with wrong frames to put it out of service. This attack has a *Safety* impact. If it succeeds, the attacked vehicle will have no brakes and driver and passengers lives will be threatened. ECU DoS attacks via malwares are avoided by using secure software updates, secure boot and disabling ECU debug interfaces. In this work, we consider ECU DoS attacks via flooding with malicious frames.

- **Impersonation attack:** consists in usurping the identity of a legitimate ECU logically or physically. That is, the attacker can impersonate as an ECU by sending frames on its behalf. Or, it can replace the legitimate ECU i.e. the hardware with a fake ECU. For example, the attacker can impersonate as the decisional ECU i.e. the Vehicle Control Unit (VCU) of Figure 1. Then, it sends braking orders to BECU.
- **Isolation attack:** consists in isolating an ECU by dropping and filtering all the frames going to or coming from it. Isolation attack is only executed by A_2. It can be catastrophic when it targets vital ECUs such as BECU.

V. CAN Intrusion Detection

Our intrusion detection method relies on legitimate ECUs to detect attacks on CAN. We assume that each ECU is embedding a Hardware Security Module (HSM). The HSM is a security processor which is dedicated for cryptographic computation and secure key storage. Our assumption is realistic as all new automotive microcontrollers contain a HSM. Freescale McKinley, Infineon AURIX or Boundary Devices Nitrogen6X and Sabrelite are examples of such microcontrollers. In the following sections, we describe our proposed extensions for CAN communication protocol. Moreover, we explain how we used HSMs to enforce CAN security.

A. CAN Set of Frames Extension

We propose to extend the set of produced frames identifiers of $\mathsf{ECU_i}$, namely $\mathsf{SetPID_i}$ with two identifiers: one for Domain_Activation frames ($\mathsf{P_i^{DA}}$) and the other for Domain_Violation frames ($\mathsf{P_i^{DV}}$). We make each legitimate $\mathsf{ECU_i}$ authenticate periodically to its frames consumers by issuing a Domain_Activation frame. Then, $\mathsf{ECU_i}$ monitors the CAN bus to check that no Data frames are being sent on its behalf by a malicious entity. That is, $\mathsf{ECU_i}$ checks if a received Data frame does not contain an $\mathsf{ID_f} \in \mathsf{SetPID_i} = \{\mathsf{P_{i1}}, \ldots, \mathsf{P_{in}}, \mathsf{P_i^{DA}}, \mathsf{P_i^{DV}}\}$. In case of attack, $\mathsf{ECU_i}$ erases the intruder frame by crushing it with an Error frame. Then, it notifies its frame consumers about the detected intrusion using a Domain_Violation frame. $\mathsf{P_i^{DA}}$ and $\mathsf{P_i^{DV}}$ must be priority identifiers.

In addition, we propose to extend the set of frames of the Vehicle Control Unit (VCU, Figure 1) with a Session_Init frame. We remind that, in our architecture, VCU is the centric equipment which manages the vehicle safety and security. Consequently, we make it bootstrap the key derivation between ECUs by sending a Session_Init frame at vehicle start-up. The key derivation process is detailed in Section V-B. For the sake of clarity, we set $\mathsf{ECU_1} = \mathsf{VCU}$. So, we add to $\mathsf{SetPID_1}$ the Session_Init frame identifier ($\mathsf{P_1^{SI}}$). As such, we obtain $\mathsf{ECU_1}$ final set of produced identifiers $\mathsf{SetPID_1} = \{\mathsf{P_{11}}, \ldots, \mathsf{P_{1n}}, \mathsf{P_1^{DA}}, \mathsf{P_1^{DV}}, \mathsf{P_1^{SI}}\}$.

B. Key Management

We associate a distinct Master Key $\mathsf{MK_i}$ to each domain formed by a frame producer $\mathsf{ECU_i}$ and its $\mathsf{SetConsumers_i}$. For Figure 6 example, $\mathsf{ECU_2}$ frame with $\mathsf{ID_f} = \mathsf{P_{21}}$ is consumed by $\mathsf{ECU_1}$ and $\mathsf{ECU_4}$. So, we associate the key $\mathsf{MK_2}$ to $\mathsf{ECU_2}$ and $\mathsf{SetConsumers_2} = \{\mathsf{ECU_1}, \mathsf{ECU_4}\}$.

From master keys, ECUs compute a temporary key (TK). The latter serves to compute MACs for Session_Init, Domain_Activation or Domain_Violation frames. We store MK and TK in a tamper-resistant memory within a HSM as proposed by Schweppe et al. [9]. In addition, we associate a hardware identification tag to all keys within the HSM. The tag distinguishes MAC computation keys (S) from MAC verification ones (V). That is, keys tagged as (S) cannot be used to check a MAC validity. Meanwhile, keys tagged as (V) cannot be used to compute a MAC. As such, a malicious $\mathsf{ECU_j}$ cannot impersonate as a frame producer $\mathsf{ECU_i}$ by sending a signed Data frame with $\mathsf{ID_f} = \mathsf{P_i^{DA}}$ to $\mathsf{SetConsumers_i}$. Indeed, $\mathsf{ECU_j}$ cannot use the temporary key of $\mathsf{ECU_j}$ tagged as (V) to compute a MAC over a forged Data frame having $\mathsf{ID_f} = \mathsf{P_i^{DA}}$.

*1) **Master Keys**:* We denote the set of master keys, stored in the HSM of $\mathsf{ECU_i}$, by $\mathsf{SetMK_i} = \{(\mathsf{MK_{i1}}, \mathsf{T_{i1}}, \mathsf{I_{i1}}), \ldots, (\mathsf{MK_{in}}, \mathsf{T_{in}}, \mathsf{I_{in}})\}$. $\mathsf{SetMK_i}$ not only contains the master keys shared by $\mathsf{ECU_i}$ with its consumers, but also holds the master keys of producers $\mathsf{ECU_{j\in[1,n],j\neq i}}$ for whom $\mathsf{ECU_i} \in \mathsf{SetConsumers_j}$. $(\mathsf{MK_{ik}}, \mathsf{T_{ik}}, \mathsf{I_{ik}})$ refers to the master key $\mathsf{MK_{ik}}$ tagged as $\mathsf{T_{ik}}$ and associated to the ECU identified by $\mathsf{I_{ik}}$. $\mathsf{MK_{ik}}$ is 256-bits long. $\mathsf{T_{ik}}$ value is either (S) or (V). $\mathsf{I_{ik}}$ is a unique 32-bits value that identifies a frame producer. $\mathsf{I_{ik}}$ is needed during $\mathsf{TK_{ik}}$ derivation from $\mathsf{MK_{ik}}$. $\mathsf{MK_{ik}}$ and $\mathsf{I_{ik}}$ values are fixed during car manufacturing.

*2) **Temporary Keys**:* We denote the set of temporary keys, stored in the HSM of $\mathsf{ECU_i}$, by $\mathsf{SetTK_i} = \{(\mathsf{TK_{i1}}, \mathsf{T_{i1}}, \mathsf{CTR_{i1}^F}, \mathsf{CTR_{i1}^A}), \ldots, (\mathsf{TK_{in}}, \mathsf{T_{in}}, \mathsf{CTR_{in}^F}, \mathsf{CTR_{in}^A})\}$. $\mathsf{TK_{ik}}$ is derived from $\mathsf{MK_{ik}}$ and inherits its tag value $\mathsf{T_{ik}}$. $\mathsf{CTR_{ik}^F}$ is a counter value that provides Domain_Activation and Domain_Violation frames freshness. Meanwhile, $\mathsf{CTR_{ik}^A}$ is an alert counter which is increased each time a Domain_Violation frame is sent signed with $\mathsf{TK_{ik}}$. Temporary keys are 256-bits long, while $\mathsf{CTR_{ik}^F}$ and $\mathsf{CTR_{ik}^A}$ are 16-bits long.

*3) **Keys Derivation**:* For the sake of simplicity, we set $(\mathsf{MK_{i1}}, \mathsf{T_{i1}}, \mathsf{I_{i1}})$ equal to $(\mathsf{MK_i}, \mathsf{S}, \mathsf{I_i})$. The latter refers to the master key shared by $\mathsf{ECU_i}$ with its frame consumers. Meanwhile, the remaining master keys $\mathsf{MK_{j\in[1,n],j\neq i}}$ belong to $\mathsf{ECU_{j\in[1,n],j\neq i}}$ for whom $\mathsf{ECU_i}$ is a consumer (i.e. $\mathsf{ECU_i} \in \mathsf{SetConsumers_j}$). These master keys are tagged as (V). For Figure 6 example, $\mathsf{ECU_1}$ stores in $\mathsf{SetMK_1}$ its own domain master key $(\mathsf{MK_1}, \mathsf{S}, \mathsf{I_1})$ and the tuples $(\mathsf{MK_2}, \mathsf{V}, \mathsf{I_2})$, $(\mathsf{MK_3}, \mathsf{V}, \mathsf{I_3})$ and $(\mathsf{MK_4}, \mathsf{V}, \mathsf{I_4})$ that correspond to $\mathsf{ECU_2}$, $\mathsf{ECU_3}$ and $\mathsf{ECU_4}$, respectively.

$\mathsf{TK_i}$ is derived from $\mathsf{MK_i}$ within the HSMs of $\mathsf{ECU_i}$ and $\mathsf{ECU_{j,j\neq i}} \in \mathsf{SetConsumers_i}$. $\mathsf{TK_i}$ is computed as $\mathsf{TK_i} = \mathsf{KDF}(\mathsf{MK_i}||\mathsf{I_i}||\mathsf{CTR_i^S}||\mathsf{N})$ where KDF is a key derivation function, $\mathsf{I_i}$ an ECU identifier, $\mathsf{CTR_i^S}$ a session counter

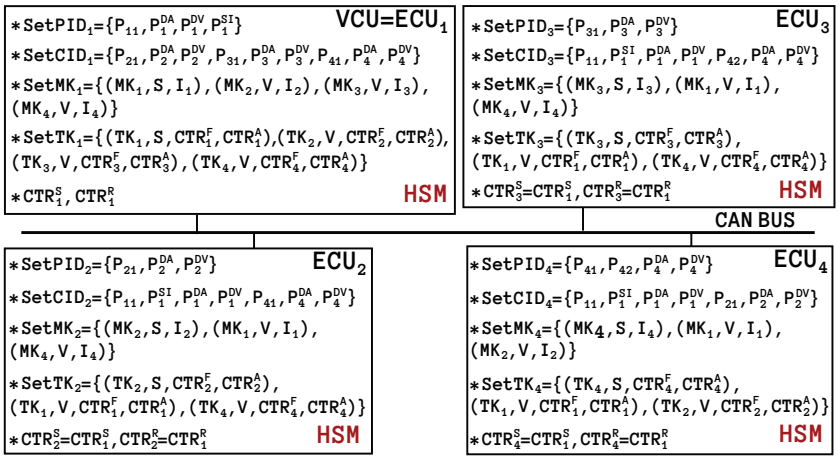

Fig. 6. Key initialization between ECUs

and N a 128-bits long nonce. KDF is a MAC computation algorithm such as HMAC [20]. CTR_i^S is a 64-bits long counter. CTR_i^S serves to keep a track of the current session number. In addition, it proves the nonce N freshness. The vehicle control unit i.e. ECU_1 provides CTR_i^S and N values to any ECU_i at vehicle *start-up* via the Session_Init frame.

*4) **Keys Bootstrapping**:* In this section, we detail the process of keys bootstrapping at vehicle start-up with respect to Figure 6 elements. First, the driver authenticates with vehicle key/card to ECU_1 using the key anti-theft protocol. The latter is secretly developed by the car manufacturer and is out-of-scope of this work. If the authentication succeeds, ECU_1 sends a *Wake-up* signal to other ECUs. The latter are kicked-off and respond with a *Ready* signal. After noticing that ECUs are started, ECU_1 sends a Session_Init frame.

Session_Init payload contains a 16-bits retransmission counter CTR_1^R initialized to 0, the session counter value CTR_1^S incremented by 1, a 128-bits nonce N and a MAC. The MAC inputs are: $ID_f = P_1^{SI}$, CTR_1^R, $CTR_1^S = CTR_1^S + 1$ and N. The MAC computation key is $TK_1 = KDF(MK_1||I_1||CTR_1^S||N)$. Session_Init frame is transmitted *fragmented* to all ECUs (ECU_2, ECU_3 and ECU_4) as its payload exceeds 64-bits.

At the reception of the Session_Init frame, ECU_2, ECU_3 and ECU_4 check that $CTR_1^R = 0$. A value of $CTR_1^R = 0$ implies that the vehicle has been just started. In addition, they verify the freshness of the received information by checking that $CTR_1^S > CTR_2^S$, $CTR_1^S > CTR_3^S$ and $CTR_1^S > CTR_4^S$. Then, they derive the temporary key $TK_1 = KDF(MK_1||I_1||CTR_1^S||N)$. If the received MAC is valid, ECU_2, ECU_3 and ECU_4 authenticate ECU_1 as the sender of the Session_Init frame. In addition, they update their respective session counters as follows: $CTR_2^S = CTR_1^S$, $CTR_3^S = CTR_1^S$ and $CTR_4^S = CTR_1^S$.

Finally, each $ECU_{i,i\in\{1,2,3,4\}}$ derives the remaining temporary keys that are shared with its frame consumers and producers. For Figure 6 example, ECU_2 computes its own temporary key $TK_2 = KDF(MK_2||I_2||CTR_2^S||N)$. TK_2 is tagged as a MAC computation key (S). However, ECU_2 derives the key shared with ECU_4 as $TK_4 = KDF(MK_4||I_4||CTR_2^S||N)$. TK_4 is tagged as a MAC verification key (V).

*5) **ECU Session Synchronization**:* It may happen that an ECU e.g. ECU_2 wakes-up after the transmission of the Session_Init frame or reboots during an on-going session due to a dysfunction. In these two cases, ECU_2 sends a delayed *Ready* signal to VCU, i.e. ECU_1. Consequently, ECU_1 retransmits a Session_Init frame. This Session_Init payload differs from the one of the previous Section V-B4. It contains an incremented value of the retransmission counter CTR_1^R while the value of the session counter CTR_1^S and the nonce N are kept unchanged. In addition, it includes a MAC computed with the same key TK_1.

At the reception of Session_Init, ECU_2 notices that $CTR_1^R > (CTR_2^R = 0)$. Consequently, it verifies that $CTR_1^S \geqslant CTR_2^S$. If $CTR_1^S = CTR_2^S$, ECU_2 understands that it has rebooted while the vehicle was functioning. Meanwhile, If $CTR_1^S > CTR_2^S$, ECU_2 understands that it has missed the initial Session_Init frame. In the two cases, ECU_2 derives TK_1 as in Section V-B4 and validates the MAC. Finally, it derives the remaining temporary keys TK_2 and TK_4. It also sets $CTR_2^R = CTR_1^R$.

At the reception of Session_Init, ECU_3 and ECU_4 notice that $CTR_3^S = CTR_1^S$ and $CTR_4^S = CTR_1^S$, respectively. Consequently, they check the validity of the received MAC using TK_1. Finally, they update their retransmission counters as follows: $CTR_3^R = CTR_1^R$ and $CTR_4^R = CTR_1^R$.

The first ECU that detects that CTR_1^R has exceeded a threshold T^R, notifies the vehicle driver about the risk of a DoS attack against ECU_1.

C. ECU Initial Authentication

Once temporary session keys have been derived, each ECU sends a Domain_Activation frame to its frame consumers. The Domain_Activation frame payload contains a counter value (CTR_f) concatenated to a MAC value. The counter field length is 16-bits. CTR_f is associated to the counter of the key needed for the MAC computation. Indeed, CTR_f is equal to the key counter incremented by 1. For Figure 6 example, ECU_2 includes $CTR_f = CTR_2^F + 1$ in its Domain_Activation frame. The MAC is computed over the arbitration, control and payload fields of the CAN frame.

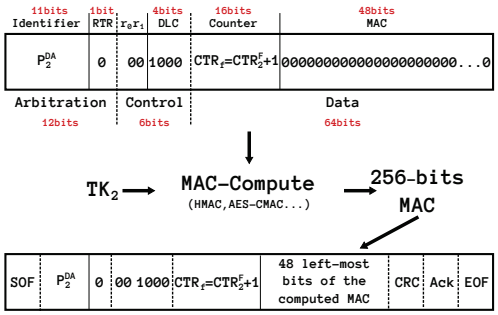

Fig. 7. ECU_2 Domain_Activation frame MAC computation

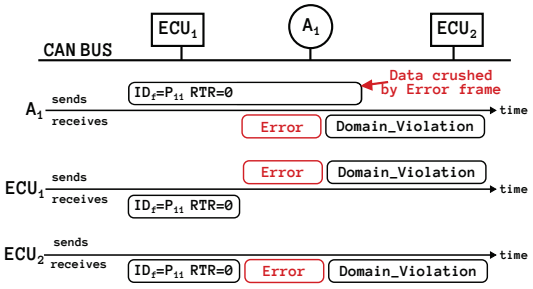

Fig. 8. ECU_1 intrusion detection

Figure 7 describes ECU_2 inputs to the MAC computation algorithm. The MAC calculus is done within the HSM. The computed MAC length is 256-bits. However, it is truncated to the 48 left-most bits in order to fit into one CAN frame.

The frame consumers (ECU_1 and ECU_4) check the freshness of ECU_2 Domain_Activation frame by comparing the frame counter value CTR_f to their local values of CTR_2^F. CTR_f must be strictly superior to CTR_2^F ($CTR_f > CTR_2^F$). Then, they check the MAC using TK_2 tagged as a MAC verification key (V) and the same algorithm of Figure 7. If the verification succeeds, they set CTR_2^F to CTR_f and they authenticate ECU_2 as the legitimate producer of the Domain_Activation frame.

D. ECU Intrusion Detection

After sending their first Domain_Activation frames, ECUs start monitoring the CAN bus to detect intrusions. Each ECU checks that the received Data frame does not contain one of its produced frame identifiers. For example, ECU_1 checks that received Data frame does not contain a $ID_f \in SetPID_1$. If $ID_f \in SetPID_1$, ECU_1 detects that a malicious entity is impersonating as itself. As a consequence, ECU_1 erases directly the attacker frame with an Error frame (Figure 8). Then, ECU_1 transmits a Domain_Violation frame to $SetConsumers_1$. The Domain_Violation frame has the same payload format as the Domain_Activation frame. That is, it contains a counter value (CTR_f) concatenated to a MAC. CTR_f corresponds to an incremented value of CTR_1^F. The MAC is computed as presented in Figure 7. In addition, ECU_1 increments the intrusion alert counter CTR_1^A.

The frame consumers (ECU_2, ECU_3 and ECU_4) check the freshness of ECU_1 Domain_Violation frame by verifying that $CTR_f > CTR_1^F$. Then, they verify the MAC value using TK_1. Finally, they set CTR_1^F to the received CTR_f and increment local counters for alerts coming from ECU_1, namely CTR_1^A . The first ECU that detects that CTR_1^A has exceeded a threshold T^A, notifies the vehicle driver about the detected intrusion. T^A limits the intrusion attempts on an ECU. At this point, the driver must take its vehicle to the garage to investigate the origin of the intrusion.

E. Security Analysis

- **DoS and impersonation attacks:** our proposed intrusion detection mechanism thwarts attacks that are executed by attacker A_1. It permits to detect network based DoS and impersonation attacks. In practice, the latter require connecting a special equipment to the CAN bus via the On-Board Diagnostics plug (OBD, Figure 1). Then, the attacker remotely controls this equipment to forge fake frames with valid identifiers that belong to a legitimate ECU. However, our proposed solution detects this fake frames and annihilates them. That is, frame consumers will not receive the payload of a malicious frame as it is erased by an Error fame followed by a Domain_Violation frame (Figure 8).
- **Replacement attacks:** correspond to hardware impersonation attacks where the attacker physically replaces a legitimate ECU by a malicious one. These attacks are detected by $SetConsumers_i$ when they do not receive a valid Domain_Activation from ECU_i.
- **Isolation attacks:** our proposed intrusion detection method does not provide countermeasure to an isolation attack which is carried by attacker A_2 (Figure 5). In fact, A_2 controls all the data flow of its target, e.g. ECU_4. Consequently, she can impersonate as ECU_4. Note that even if other intrusion detection methods such as the correlation of signals fingerprints are used, A_2 can still drop packets going to or coming from ECU_4 without being noticed. In practice, isolation attack requires a physical access to the CAN bus. The attacker have to cut the wires in order to interpose its filtering equipment in front of the targeted ECU. This operation is not trivial and requires the knowledge of the detailed car embedded architecture.
- **Replay attacks:** our proposed intrusion detection avoids replay attacks. Replayed frames are detected, by the frame legitimate producer, as coming from an intruder. In addition, we rely on counters to prove the freshness of Domain_Violation, Domain_Activation and Session_Init frames. In a Session_Init frame, the nonce N is associated to the session counter CTR_1^S that is maintained by VCU (ECU_1 in Figure 6). CTR_1^S is 64-bits long. Consequently, for a number of start-up equal to $32 = 2^5$ times per day, the lifetime of a vehicle is around $2^{64-5} = 2^{59}$ days.
- **Key renewal:** temporary keys are renewed every time the vehicle is started. In addition, their associated counters are re-initialized to 0. As such, each ECU can send $2^{16} - 1$ signed Domain_Violation or Domain_Activation frames. If no intrusion is detected, each ECU sends periodically a Domain_Activation frame to its consumers to prove that it is still alive. For a period equal to 2 seconds, a vehicle can run for $(2^{16} - 1)/1800 \simeq 36$ hours with no need for key refreshing. That is, an attacker will have 36 hours

to guess, i.e. brute force, a 256-bits long temporary key in order to impersonate as a legitimate ECU by sending valid Domain_Violation or Domain_Activation frames. Let us consider that the attacker has a machine with 1024 cores each cadenced at 4GHz. If we suppose that a new key is tried every CPU clock cycle, this attacker needs 8.964×10^{56} years to find the good key.

- **MAC truncation:** 48-bits is a sufficient length for MAC truncation [21]. Indeed, we cannot afford to use more than one CAN frame to transmit the MAC value due to CAN reduced bandwidth (at most 1Mbits/s) and real-time constraint. If we allow 2^8 failed MAC verifications, the likelihood of accepting forged data will be $\frac{1}{2^{40}}$ per temporary key [21]. Of course, the MAC temporary keys have to be changed before reaching 2^8 MAC failed verifications.

VI. Conclusion

We describe in this work a simple intrusion detection method for controller area networks. Our proposed solution relies on CAN legitimate controllers for the detection of intruders. At vehicle starting, each ECU authenticates to other ECUs by sending a Domain_Activation frame. Then, each ECU monitors the CAN bus to detect whether an attacker is sending frames on its behalf or not. Our solution detects ECU DoS, replay and impersonation attacks. The proposed intrusion detection is also relevant to CAN FD [22] which extends CAN 8-bytes payload to 64-bytes. The use of a larger payload is advantageous as we can include larger counter and the full MAC in Domain_Activation and Domain_Violation frames. That is, session keys will last longer as their associated counters can be increased from 2-bytes to 8-bytes, for example. In addition, we do not need MAC truncation with CAN FD.

Our future work consists in improving this intrusion detection system to thwart isolation attacks. In addition, we will work on improving the key initialization scheme to suit situation where ECUs have been accidentally restarted or need to be changed/removed from vehicles.

ACKNOWLEDGMENT

This work has been carried out in *IRT SystemX*, and therefore granted with public funds within the scope of the French program *Investissements d'avenir*. This work is part of Automotive Electronics and Software project.

References

[1] Bosch, "CAN Specification Version 2.0," Stuttgart, September 1991.

[2] T. Hoppe, S. Kiltz, and J. Dittmann, "Security Threats to Automotive CAN Networks - Practical Examples and Selected Short-Term Countermeasures," in *Proceedings of the 27th International Conference on Computer Safety, Reliability, and Security (SAFECOMP 2008)*.

[3] D. K. Nilsson and U. E. Larson, "Simulated Attacks on CAN Buses: Vehicle Virus," in *Proceedings of the Fifth International Conference on Communication Systems and Networks (AsiaCSN 2008)*.

[4] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, and S. Savage, "Experimental Security Analysis of a Modern Automobile," in *Proceedings of the 2010 IEEE Symposium on Security and Privacy*.

[5] S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, K. Koscher, A. Czeskis, F. Roesner, and T. Kohno, "Comprehensive Experimental Analyses of Automotive Attack Surfaces," in *Proceedings of the 20th USENIX Conference on Security*.

[6] D. Schneider, "Jeep hacking 101." [Online]. Available: http://spectrum.ieee.org/cars-that-think/transportation/systems/jeep-hacking-101

[7] H. Oguma, A. Yoshioka, M. Nishikawa, R. Shigetomi, A. Otsuka, and H. Imai, "New Attestation Based Security Architecture for In-Vehicle Communication," in *IEEE Global Telecommunications Conference (GLOBECOM 2008)*.

[8] C. Szilagyi and P. Koopman, "Flexible multicast authentication for time-triggered embedded control network applications," in *2009 IEEE/IFIP International Conference on Dependable Systems Networks*.

[9] H. Schweppe, Y. Roudier, B. Weyl, L. Apvrille, and D. Scheuermann, "Car2X communication: securing the last meter -A cost-effective approach for ensuring trust in Car2X applications using in-vehicle symmetric cryptography," in *4th IEEE International Symposium on Wireless Vehicular Communications (WiVeC'2011)*.

[10] C.-W. Lin and A. Sangiovanni-Vincentelli, "Cyber-Security for the Controller Area Network (CAN) Communication Protocol," in *2012 International Conference on Cyber Security (CyberSecurity 2012)*.

[11] B. Groza, S. Murvay, A. van Herrewege, and I. Verbauwhede, "LiBrA-CAN: A Lightweight Broadcast Authentication Protocol for Controller Area Networks," in *Cryptology and Network Security*, ser. Lecture Notes in Computer Science, J. Pieprzyk, A.-R. Sadeghi, and M. Manulis, Eds., vol. 7712. Springer Berlin Heidelberg, 2012, pp. 185–200.

[12] B. Groza and S. Murvay, "Efficient Protocols for Secure Broadcast in Controller Area Networks," *IEEE Transactions on Industrial Informatics*, vol. 9, no. 4, pp. 2034–2042, Nov 2013.

[13] A. Perrig, R. Canetti, D. Song, and J. D. Tygar, "Efficient and Secure Source Authentication for Multicast," in *2001 Network and Distributed System Security Symposium*, 2001, pp. 35–46.

[14] "Security threats to automotive CAN networks—Practical examples and selected short-term countermeasures," *Reliability Engineering & System Safety*, vol. 96, no. 1, pp. 11 – 25, 2011.

[15] P. Kleberger, T. Olovsson, and E. Jonsson, "Security aspects of the in-vehicle network in the connected car," in *2011 IEEE Intelligent Vehicles Symposium*, June 2011, pp. 528–533.

[16] U. Larson, D. Nilsson, and E. Jonsson, "An approach to specification-based attack detection for in-vehicle networks," in *2008 IEEE Intelligent Vehicles Symposium*, June 2008, pp. 220–225.

[17] T. Hoppe, S. Kiltz, and J. Dittmann, "Adaptive Dynamic Reaction to Automotive IT Security Incidents Using Multimedia Car Environment," in *Fourth International Conference on Information Assurance and Security (ISA 2008)*.

[18] D. Dolev and A. Yao, "On the security of public key protocols," in *IEEE Transactions on Information Theory*, vol. 29, no. 2, March 1983.

[19] A. Boudguiga, A. Boulanger, P. Chiron, W. Klaudel, H. Labiod, and J.-C. Seguy, "RACE: Risk analysis for cooperative engines," in *7th International Conference on New Technologies, Mobility and Security (NTMS 2015)*.

[20] C. Lily, "SP 800-108. Recommendation for Key Derivation Using Pseudorandom Functions," Gaithersburg, MD, United States, Tech. Rep., 2009.

[21] Q. H. Dang, "SP 800-107. Recommendation for Applications Using Approved Hash Algorithms," Gaithersburg, MD, United States, 2012.

[22] Bosch, "CAN FD Specification Version 1.0," Stuttgart, April 2012.