Highly Efficient Authentication for CAN by Identifier Reallocation with Ordered CMACs

Bogdan Groza, Lucian Popa and Pal-Stefan Murvay

***Abstract*—Most of the existing works on securing the CAN bus are using the limited data-field of CAN frames to embed a cryptographic payload. Only very few works have suggested the use of the identifier field since identifiers are critical for the arbitration procedure and changing them at random would interfere with message priorities. To preserve priority on the bus, in this work we use an ordered CMAC buffer. In this way, we can authenticate the identifiers of CAN frames and check that the sender is a legitimate node while arbitration on the bus remains unaltered. Moreover, we determine that for real-world scenarios the achieved security level is very close to the length of the ID field despite the constraints from ordering. This procedure easily circumvents replay attacks and fuzz testing on the bus, which were exploited by many recent works. We prove the feasibility of our approach by testing practical implementations on automotive-grade microcontrollers and CAN-bus traffic allocations from a high-end vehicle. The computational requirements are some of the lowest achievable for securing CAN, with a dozen CMAC-AES computations being sufficient for extracting a table of one hundred identifiers.**

I. Introduction and motivation

The Controller Area Network (CAN) is a bus designed by BOSCH in the 80's. Special care was taken to ensure its reliability and keeping its cost low, but there were no intentions in adding security at the time of its design. Hence, a plethora of attacks on modern vehicles were recently reported, e.g., [5], [21], most of them due to the insecurity of the CAN bus. Embedding cryptographic payloads in CAN data-fields is one of the most common approaches for securing the CAN bus. This procedure is pursued by many works, e.g., [13], [29], [30], [3], etc., as well as by recent industry recommendations [2]; a brief overview can be found in [10].

Current standards [2] recommend 24-28 bits for the cryptographic authentication tag computed via a MAC (Message Authentication Code), the instantiation recommended by the standard is CMAC-AES, and 0–8 bits for the freshness parameter (according to the Security Profiles in chapter 7.10 from [2]). This is not much, and arguably it is insufficient for security needs, but it is all that that can be embedded without compromising more than 50% of the data-field. Not surprising, there are several approaches that try to optimize signal allocation in CAN frames in order to optimize the allocation of security bits, e.g., [18], [19], [33]. Recent approaches have also tried to limit the use of the data-field by using time-covert authentication channels but the security

Bogdan Groza, Lucian Popa and Pal-Stefan Murvay are with the Faculty of Automatics and Computers, Politehnica University of Timisoara, Romania, Email: bogdan.groza@aut.upt.ro, lucian.popa.lp@gmail.com, pal-stefan.murvay@aut.upt.ro

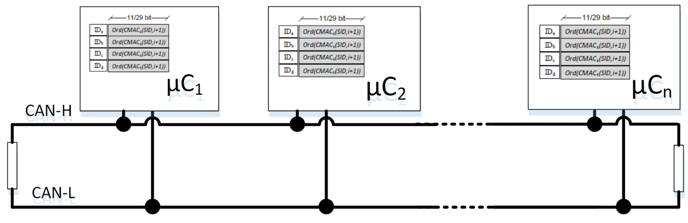
11/29 bit
ID$_a$ Ord(CMAC$_k$(SID,i+1))
ID$_b$ Ord(CMAC$_k$(SID,i+1))
ID$_c$ Ord(CMAC$_k$(SID,i+1))
ID$_d$ Ord(CMAC$_k$(SID,i+1))
µC$_1$
µC$_2$
µC$_n$
CAN-H
CAN-L

Fig. 1. Basic depiction of the CAN bus

level is even more drastically limited to 1 or several bits per frame [11], [35] which is insufficient for current needs. Similar problems related to a security level that is hard to determine are drawback for proposals that rely on physical signal characteristics [7], [8], [16] or network delays [6] (for example clock-skews can be faked as shown in [26], while voltage characteristics may be even more prone to unpredictable changes in bus impedance due to temperature, adding additional nodes on the bus, etc.). Thus relying on cryptographic authentication seems the only clear way to get a reliable security bound.

Figure 1 shows a basic depiction of the CAN bus, which is a two wire broadcast bus. In the depiction, each ECU is in possession of a cryptographically generated ID table on which we will discuss more later. The main problem in securing CAN comes from the data-field of CAN frames which is limited to 64 bits (even more, most of these bits are already in use in practical implementations). The CAN frame begins with a start of frame (SOF), it has an identifier (ID) of 11 (or 29 bits in extended frames) used in the arbitration procedure, a control field (CTL), a data-field of up to 64 bits, a CRC, an acknowledgment bit (ACK) and finally the end-of-frame (EOF) which marks the end of the frame.

In this work we exploit the identifiers of CAN frames and use them as an authentication tag to identify senders. This does not exclude regular authentication of the data-field, mandated by recent standards, e.g., [2], but rather complements it. Such a procedure garners at least two advantages. First, as the data-field carries only a limited amount of authentication bits, by using all the bits from the ID (as additional security bits) we immediately increase the security level of data carried by CAN frames. Second, by essentially encrypting the ID, we keep the IDs hidden from a malicious adversary thus keeping the sender anonymous. Notably, most standardized CAN-based higher-layer protocols do encode the identity of senders in the ID field. The ISO 15765-2 protocol, also known as ISO-

TP, uses the CAN frame ID to transmit either the address of the message source or a unique identifier associated to the transmitter-receiver pair [15]. Some of the most commonly known uses of ISO-TP is for message transmission using the UDS (Unified Diagnostic Serviced) and OBD-2 (On-Board Diagnostics) diagnostic protocols. A similar approach is employed by the J1939 family of protocols, used in commercial vehicles (i.e. trucks, buses, agricultural machines and even marine navigation), in which the message source address is encoded as part of the ID field [25]. According to the J1939 specification, each node/functional unit is uniquely assigned a source address to enable transmitter identification. Other CAN-based protocols employed in industrial control networks also use the frame ID for sender identification. One such example is the DeviceNet protocol which identifies transmitter nodes by so-called MAC IDs transmitted in the identifier field [24]. Thus, encoding sender identification information in the ID leaves CAN vulnerable to privacy attacks as the sender of the message can be identified by adversaries. Consequently, the protocol that we propose also helps in keeping the sender anonymous in front of eavesdroppers and the IDs can be uniquely re-mapped to their original values by genuine nodes from the network (allowing only genuine nodes to correctly identify senders).

To keep arbitration intact on the bus, we use a sorted CMAC-AES buffer, which is in principle a form of order-preserving encryption, a well-known and understood cryptographic primitive, see [1], [4]. Rather than using an encryption function however, we use a Message Authentication Code (MAC), i.e., CMAC-AES, for compliance with existing standards in automotive security, e.g., [2]. The extension from order-preserving encryption to order-preserving MACs is however straight-forward since mutatis-mutandis both encryptions and MACs are cryptographic one-way functions. That is, rather than using the output of an encryption function $E_k(m)$ we use the output of a MAC function $MAC_k(m)$ (here k stands for a symmetric key and m for a message). To save on computational time we split the CMAC outputs in arrays of 11 or 29 bits (according to the identifier length). If computational power is not a main limitation, e.g., if hardware support is available, one can use a single CMAC computation for each ID.

A. *Proposal overview and related works*

Figures 2 and 3 give an overview of the result. We preserved the periodicity of the original IDs recorded in a high-end vehicle, then re-mapped and re-constructed the traffic on our laboratory setup. Figure 2 shows the re-map when working in the 11-bit ID space. The left picture shows a higher agglomeration of IDs in the lower side. This is expected since higher priority IDs are sent more often on the bus and accumulate faster. This will however decrease the entropy of high priority IDs making them less secure. On the right side of Figure 2 we show the result after applying one small bias in the probability distribution that allows more space for high priority IDs (the lower part of the figure). In this way, high priority IDs will get a better security level. More details on this will be given in the experimental section. Figure 3 shows the allocation in the 29-bit identifier space. Part (i) of the figure shows the allocation of all 88 IDs, this looks more random compared to the 11-bit space as there is more space to allocate the IDs. Part (ii) of the figure is easier to interpret. Here we show only the evolution of 6 of the IDs (there are 2 IDs in the lower end that are not fully visible). While the values of the IDs are changed at random each $100ms$, note that the order between the IDs is still preserved. Subsequently, in parts (iii) and (iv) we show the effects of applying a 1-bit and 2-bit bias. This leaves more room for high priority IDs which are lower valued and allocated in the lower part of the distribution as depicted in the image.

Only very few works have focused on using the ID field for authentication. Perhaps the earliest of them is [12]. Three more recent works that have also investigated the use of the identifier fields to enhance the security of CAN are [14], [32] and [31]. Interestingly, none of these works calls for the relation with order-preserving cryptographic functions. Moreover, [14] focuses on the ID-hopping technique as a mean to prevent DoS attacks rather than to assure authenticity of the senders. In both [12] and [31] additional bits from the ID are used to keep arbitration intact, this is distinct from our procedures which take advantage of the entire ID field and are based on the output of an AES-CMAC function. The work in [32] is very close to ours but the main difference is that [32] relies on a more complex hardware implementation, we keep our entire implementation in software and show that it is quite inexpensive in computational terms. Our main contribution is in providing a faster solution, that can be efficiently implemented at the software layer with the help of binary sorted trees or doubly linked lists, to generate ordered AES-CMACs by which the original ID table can be efficiently remapped. The proposed solution requires only several CMAC-AES computations and a binary sorted tree for generating a fresh ID table. As we show in the experimental section this can be done in essentially hundreds of microseconds for a high number of IDs, in this way securing a larger communication cycle (hundreds of milliseconds or even seconds according to specific needs). Moreover, [32] analyzes the security of their solution in terms of Shannon entropy. As we later discuss the minimum entropy is the desired metric for establishing a security bound as Shannon entropy may be somewhat misleading as it gives a higher value than the actual security level in front of a guessing adversary. We also contribute with a detailed analysis of the experimental data, based on well-known security metrics such as guessing probability and min entropy, and provide computational and bandwidth results on automotive-grade controllers for real-world in-vehicle traffic. We add two more flavours to our solution: destroying illegitimate frames on-the-fly by fast searching for genuine IDs in a binary sorted tree and an efficient solution for biasing the ID distribution such that higher priority IDs will get a higher entropy.

Briefly, our protocol, which we'll now generically name CAN-TORO, i.e., *CAN Transmitter authentication by ORdered One-way functions*, and later discuss explicit CMAC-AES instantiations on it, has the following advantages:

- highly efficient in terms of computational requirements

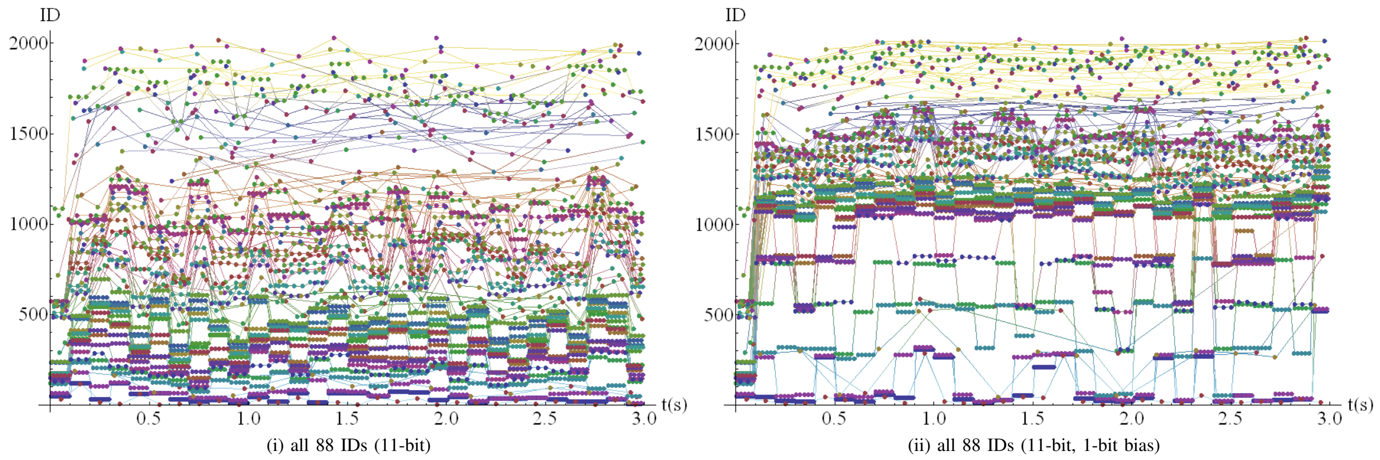

Fig. 2. View of ID re-mapping in the 11-bit space during the first 3s of runtime (IDs updated at $100ms$)

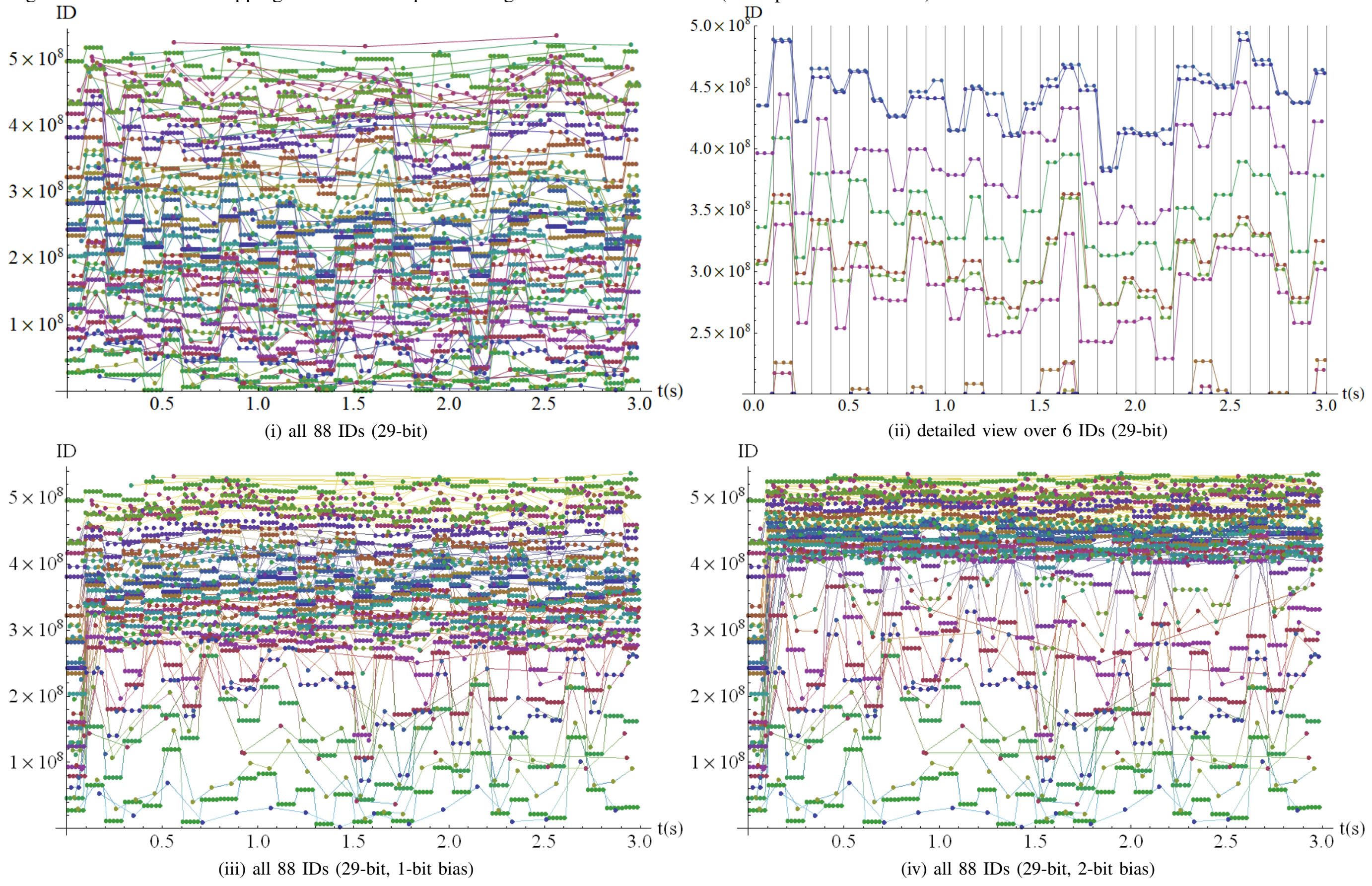

Fig. 3. View of ID re-mapping in the 29-bit space during the first 3s of runtime (IDs updated at $100ms$)

(e.g., less than a dozen CMAC-AES computations for an ID table of 100 IDs),

- very easy to implement in software, it calls only for standard CMACs and a binary sorted tree (alternatively, with small performance penalties, we also use a doubly-linked list to avoid recursion lowering stack requirements),
- flexible in implementation, any cryptographic one-way function is suitable for the computation of the ID table (we keep the AES-CMACs instantiation to align our work with existing standards),
- requires no bits from the data-field of CAN frames which is already limited to 64 bits,
- frame priority is kept unchanged despite randomization of the ID field in CAN frames,
- security level is close to the maximum number of bits in the ID field, despite preserving order in the freshly generated ID table,
- the entropy (which is the security level) of the IDs can be easily biased (by simply using bit masks) to give higher priority IDs an increased entropy,
- illegitimate IDs can be detected on-the-fly and the corresponding frame destroyed before completion by legitimate nodes (this is facilitated by a quick search in a binary sorted tree of IDs which is performed in less than

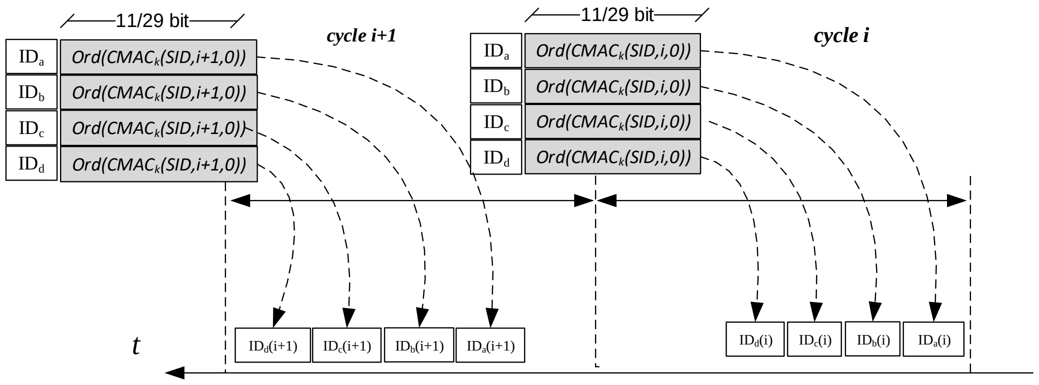

Fig. 4. ID reallocation with order preserving CMACs and arrival on the bus

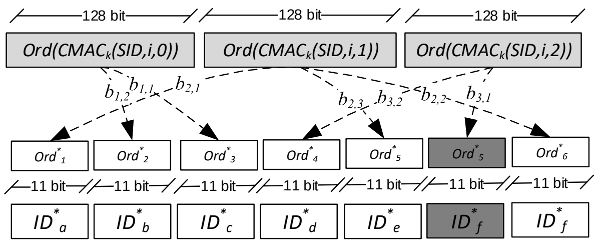
128 bit
Ord(CMACk(SID,i,0))
Ord(CMACk(SID,i,1))
Ord(CMACk(SID,i,2))
b1,2 b1,1 b2,1 b2,3 b3,2 b2,2 b3,1
Ord*1 Ord*2 Ord*3 Ord*4 Ord*5 Ord*5 Ord*6
11 bit
ID*a ID*b ID*c ID*d ID*e ID*f ID*f

Fig. 5. Overview of ID reallocation by ordered CMACs

a micro-second by high-end controllers in our setup).

Of course, no protocol is without shortcomings. As we later discuss, an adversary that has access to the bus and can unplug nodes may use correctly generated IDs to inject frames. To avoid this, regular authentication of the data-field as per AUTOSAR standard [2] cannot be excluded. However, it is worth noting that this type of adversary will require physical access to the bus and the ability to perform physical modifications of the bus topology, which is less convenient. Replay of existing IDs is possible if the ID table is not refreshed too often, a reason for which we also recommend intrusion-detection systems based on frame periodicity (e.g., [28], [22], [11] or any other type of IDS, e.g., [34], [23]) to be in place. It would be out-of-scope for the current work to merge all these approaches.

II. Concept and protocol design

In this section we illustrate the main concept then we give specific details on the protocol that we design and implement.

A. Concept in a nutshell

We assume that communication over the CAN bus is done in cycles of duration Δ, this is consistent with real-world deployments since most of the CAN frames have a fixed periodicity. If this is not the case, for on-event frames, the IDs will be mapped based on the same procedures and along with the cyclic IDs. The IDs of on-event frames will simply remain unused in communication cycles where the on-event frame does not occur. The IDs are to be shifted/changed in each communication cycle. For practical reasons, we consider the cycle length Δ to be set to $100ms$ - $1s$, more discussions on this follow in the experimental section. In each cycle, the new IDs are derived via an order-preserving CMAC. Any other cryptographic primitive can be used for this purpose. The values of the original IDs, stored in an ID table, are to be mapped to fresh values stored in another ID table.

This concept is suggested in Figure 4. We consider four IDs, i.e., $\mathsf{ID}_\alpha, \alpha \in \{a, b, c, d\}$, in descending order of their priority, i.e., ID_a has the highest priority. The IDs arrive on the bus in the same order both in cycle i and cycle $i+1$. This happens even if they are sent exactly at the same time, i.e., a collision which is solved by the CAN arbitration mechanism, since the ordered CMACs by which they are mapped to the new identifiers will return IDs that are in identical order.

Figure 5 explains how we compute the order-preserving CMAC buffer in session i. To implement an order-preserving CMACs, we split the blocks that are output by CMAC in 11-bit blocks (or 29-bits in case of extended identifiers), we eliminate all duplicates and sort them in ascending order. In this depiction, we consider the case when a single output of the CMAC construction (128-bit in length for CMAC-AES) is not sufficient to generate all IDs and we simply compute the function three times by adding a counter $i = 0, 1, 2$, i.e., $\mathsf{CMAC}_k(\mathsf{SID}, i, 0)$, $\mathsf{CMAC}_k(\mathsf{SID}, i, 1)$, $\mathsf{CMAC}_k(\mathsf{SID}, i, 2)$ (here we use SID to denote the session identifier which is a random value to avoid replay of the same communication cycles). This is the approach from our practical instantiation in the experimental section where we use a counter that allows us to generate any number of IDs. In Figure 5 two 11-bit blocks of the first CMAC, i.e, $b_{1,2}, b_{1,1}$, will be the new values for ID^*_b and ID^*_c (here $*$ is a placeholder to denote this is a fresh ID). The first 11-bit block of the second CMAC, i.e., $b_{2,1}$ is the new value for ID^*_a, etc. For ID^*_f, block $b_{3,1}$ has collided with

$b_{2,3}$ and it is discarded while $b_{2,2}$ is kept for ID_f^*. Note that the IDs at the bottom of the figure are kept in their original order $a, b, c..., f$ - it is only the output of the CMACs that is sorted (after being split in 11-bit blocks).

B. Algorithms for ordered authentication of the identifiers

To build the ID table, we call for three algorithms that are outlined in Figures 6 and 7 and 8. We now explain these algorithms in detail.

Algorithm 1 in Figure 6, extracts identifiers of length $idlen$ from the buffer. First, in line 2, the algorithm checks if there are enough bits in the buffer and if not it updates the crypto-buffer accordingly, i.e., $OWF(k, \mathsf{SID}, counter)$ is appended to the buffer by the call to $update\_buffer()$. For AUTOSAR compliance, in our implementation we use the CMAC-AES (and later as a speed-up a hash function) for the instantiation of the one-way function. Then, the algorithm takes the current position in the buffer pos and builds two indexes, i.e., the left and right indexes $lindex$, $rindex$. If these are the same, then $idlen$ bits are extracted only from the left position (the right position is the same) otherwise $lcount$ bits are extracted from the left and $rcount$ bits from the right. This is done by the branch on line 5. Then the current position in the buffer pos is incremented with the length of the ID $idlen$ in line 16.

Algorithm 2 in Figure 7 is responsible for sorting the extracted IDs. This is done by using a binary search tree. The algorithm instantiates the first ID with a random value (line 2), then it loops for the remaining $n-1$ IDs, each time generating a new ID and placing it to the left or right of a leaf node (the branches in lines 10 and 12). If the generated ID already exists in the tree, then it will be regenerated (the branch in line 13 deals with this situation). We skip more details on Algorithm 2 since binary search trees are a well known structure in programming. Finally Algorithm 3 in the same figure performs a depth-first search on the binary tree in order to extract the IDs in an ID table. This is done by recursion on the left branch (line 2), followed by extraction of the ID then by recursion on the right branch (line 4).

To avoid recursion (which may cause problems related to stack size), alternatively, we have also used a doubly linked list implementation. The runtime is somewhat slower compared to the binary sorted approach, but it does not require recursion when extracting the IDs (these are already sorted in the doubly linked list). The algorithm is presented in Figure 8. Algorithm 4 simply loops for all of the n IDs (line 2), each time generating fresh bits for the current ID. Then in line 9 it parses the list until the current ID is larger or equal to the newly generated ID (or the end of the list is reached). If it is equal (line 12) then a new value must be selected. Otherwise, the newly generated ID is inserted before the first value that was larger than it (possibly changing the first ID in the list in line 20). If the end of the list is reached and no ID larger than it was found (line 24) then the new ID is simply added at the end of the list.

We now discuss the expected run time of the ID allocation algorithm. The algorithm loops through the output of a one-way function which can be modelled as a random oracle. The probability of selecting an i-th ID that collides with any of the rest of $i-1$ distinct IDs is: $p_{col} = \frac{i-1}{\sigma}$. Where σ is the total number of IDs, i.e., $\sigma \in \{2^{11}, 2^{29}\}$. Consequently, the average runtime of the identifier selection algorithm can be approximated as:

Algorithm 1 Identifier bit extraction (32-bit architecture)

```
1: procedure NEXTBITS()
2:     while $buflen*32 < idlen+pos+1$ do $update\_buffer(cryptobuf)$
3:     end while
4:     $lindex \leftarrow pos/32$
5:     $rindex \leftarrow (pos + idlen)/32$
6:     if $lindex = rindex$ then
7:         $lcount \leftarrow pos\%32$
8:         $val \leftarrow cryptobuf[lindex] \gg (32-idlen-lcount)$
9:         $val \leftarrow val\&((1 \ll idlen)-1)$
10:    else
11:        $lcount \leftarrow pos\%32$
12:        $rcount \leftarrow (pos + idlen)\%32$
13:        $val \leftarrow cryptobuf[lindex]\&((1 \ll 32-lcount)-1)$
14:        $aux \leftarrow cryptobuf[rindex] \gg (64-idlen+lcount)$
15:        $aux \leftarrow aux\&((1 \ll (idlen-(32-lcount)))-1)$
16:        $val \leftarrow (val \ll (idlen-32+lcount))|aux$
17:    $pos \leftarrow pos + idlen$
18:    return $val$
19: end procedure
```

Fig. 6. Algorithm for extraction of identifier bits

$$T_{av} = \sum_{i=1}^{n} \left[\sum_{j=1}^{\infty} j \left(1 - \frac{i-1}{\sigma}\right) \left(\frac{i-1}{\sigma}\right)^{j-1} \right] \quad (1)$$

This relation sums over all the identifier space, i.e., $i = 1..n$. For each ID it sums over the number of steps j multiplied by the probability to get no collision in the current step, i.e., $\left(1 - \frac{i-1}{\sigma}\right)$, times the probability to get a collision with the previous $i-1$ identifier values in all previous $j-1$ steps, i.e., $\left(\frac{i-1}{\sigma}\right)^{j-1}$. For practical purposes, in our implementation with ≈ 100 IDs in an 11-bit ID space there are roughly 2-3 selections which should be repeated as they result in collisions. This results in an average of 103 steps for 100 IDs and since the output of CMAC-AES is of 128 bits, if we cut slices of 11 bits, it all reduces to 9 AES computations. For 29-bit identifiers, the number of expected collisions is close to 0, but due to the larger ID field about 22 AES computations are required. These requirements are quite low given that an AES computations cost in the order of dozen micro-seconds. Experimental values on the runtime are given in the next section.

C. Protocol design

We assume that all controllers $\mathsf{ECU}_l, l = 1..n$ are in possession of a master secret key k which is used for cryptographic authentication and for computing the identifier table. We formally define our protocol entitled CAN-TORO, i.e., *CAN Transmitter authentication by ORdered One-way functions* as the following set of actions for each ECU:

1) $\mathsf{UpdateID}(\mathsf{T}, \mathsf{SID})$ is the procedure triggered on each node $\mathsf{ECU}_j, j = 1..n$ during cycle $i-1$ to update the

Algorithm 2 Building the ordered identifier table

```
1: procedure BUILDIDTABLE()
2:     $first \leftarrow new(ID)$; $first.val \leftarrow NextBits()$
3:     for $i = 1, i < n, i \leftarrow i + 1$ do
4:         $thisid \leftarrow new(ID)$
5:         repeat
6:             $thisid.val \leftarrow NextBits()$
7:             $isnew \leftarrow true$; $aux \leftarrow first$
8:             repeat
9:                 $last \leftarrow aux$
10:                if $aux.val > thisid.val$ then $aux \leftarrow aux.LID$
11:                else
12:                    if $aux.val < thisid.val$ then $aux \leftarrow aux.RID$
13:                    else $isnew \leftarrow false$
14:            until $isnew = false \vee aux = \perp$
15:            if $aux = \perp$ then
16:                if $last.val > thisid.val$ then $last.LID \leftarrow thisid$
17:                else $last.RID \leftarrow thisid$
18:        until $isnew = true$
19:    end for
20: end procedure
```

Algorithm 3 Extract IDs

```
1: procedure EXTRACTIDTABLE($thisid$)
2:     if $thisid.LID \neq \perp$ then $ExtractIDTable(thisid.LID)$
3:     $IDTable[index] \leftarrow thisid.val$; $index \leftarrow index + 1$
4:     if $thisid.RID \neq \perp$ then $ExtractIDTable(thisid.RID)$
5: end procedure
```

Fig. 7. Algorithms for building the identifier table by sorting identifiers in a binary tree and extraction by depth-first recursion

Algorithm 4 Building the ordered identifier table

```
1: procedure BUILDIDTABLE()
2:     for $i = 0, i < n, i \leftarrow i + 1$ do
3:         $thisid \leftarrow new()$
4:         $thisid.LID \leftarrow \perp$
5:         $thisid.RID \leftarrow \perp$
6:         $thisid.val \leftarrow NextBits()$
7:         $isnew \leftarrow true$
8:         $aux \leftarrow first$
9:         while $aux.val < thisid.val \wedge aux.RID \neq \perp$ do
10:            $aux = aux.RID$
11:        end while
12:        if $aux.val = thisid.val$ then
13:            $isnew \leftarrow false$
14:        else
15:            if $aux.val > thisid.val$ then
16:                $auxlid \leftarrow aux.LID$
17:                $thisid.LID \leftarrow aux.LID$
18:                $aux.LID \leftarrow thisid$
19:                $thisid.RID \leftarrow aux$
20:                if $aux.LID = \perp$ then
21:                    $first \leftarrow thisid$
22:                else
23:                    $auxlid.RID \leftarrow thisid$
24:            else
25:                $thisid.LID = aux$
26:                $aux.RID = thisid$
27:    end for
28: end procedure
```

Fig. 8. Algorithms for building the identifier table by sorting identifiers in doubly linked list (avoiding recursion)

ID table for the forthcoming communication cycle i, i.e., triggered at cycle Δ_i, by running Algorithm 3 in Figure 7,

2) $\mathsf{SendID}(\mathsf{ID}_j, \mathrm{m})$ is the procedure triggered at cycles δ_j (here δ_j is the cycle at which ID_j is sent) on a sender node $\mathsf{ECU}_l, l \in [1..n]$ for a frame with identifier field ID_j at which ECU_l retrieves the authenticated identifier ID_j^* and broadcasts message $(\mathsf{ID}_j^*, \mathrm{m})$ on the bus,
3) $\mathsf{ReceiveID}(\mathsf{ID}_j^*, \mathrm{m})$ at which each receiver $\mathsf{ECU}_l, l = 1..n$ upon receiving a CAN frame containing $\mathsf{ID}_j^*, \mathrm{m}$ verifies that ID_j^* exists in the authenticated identifier table and accepts the frame if so, otherwise reports an intrusion.

We also assume that a $\mathsf{SendSync}(i, \mathsf{SID}, \Delta)$ procedure exists by which a master node signals a new cycle on the bus by sending a default packet with the data-field containing current cycle number SID at cycles Δ. We assume a default ID for the frame carrying this data and a regular MAC-based authentication of this frame to avoid adversarial interventions. Due to the non-destructive arbitration of CAN, more ECUs can send the SendSync message on the bus at the same time. As we later discuss in the experimental section, the procedures required by UpdateID to update the ID table are called sequentially for part of the IDs during the inter-frame space (IFS) when a sender node is not busy. Frames with IDs that do not comply with the IDs from the table can be destroyed by error flags, we skip formalism for this step but provide experimental results later.

D. Security discussion

The proposed mechanism for authenticating the identifiers of CAN frames offers security in front of injection attacks, i.e., adversaries that plug on the bus and inject malicious frames, which are the most common attacks on the CAN bus. If a frame ID is sent more than once in a communication cycle, then an adversary may record the genuine ID and replay it. Such an attack may be easily traced by an Intrusion Detection System (IDS) that accounts for frame periodicity. There are many lines of work dedicated to this, e.g., [28], [22], [11], and such an approach is out of scope for the current paper. Our protocol may also directly circumvent such attacks by reducing the length of the communication cycles. Finally, the proposed mechanism can be circumvented if nodes are physically unplugged and an adversary acts as a man-in-the-middle by using the genuine identifiers to inject frames with its own content (this is a shortcoming for related solutions that authenticated the ID alone or only the senders). However, this attack can be easily circumvented by following the aforementioned AUTOSAR specification [2] that demands 24-28 bits for authentication and freshness parameter. Our solution does not exclude this mechanism but complements it by adding more security in the ID field. While we do not specifically evaluate the authentication of the data-field as per AUTOSAR specification [2], our experiments from the forthcoming section also present performance results on several automotive-grade controllers. In particular, for Infineon TC224 and TC297, the AES-CMAC computation is in the order of $16 - 55\mu s$ which is a modest computational requirement. Our

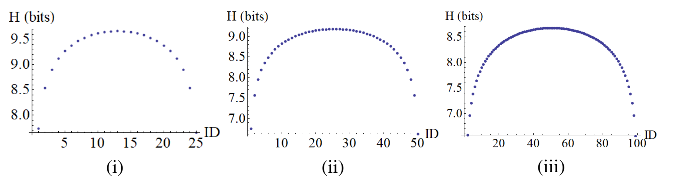

Fig. 9. Entropy variation for 11-bit IDs: effects of priority on 25 (i), 50 (ii) and 100 IDs (iii)

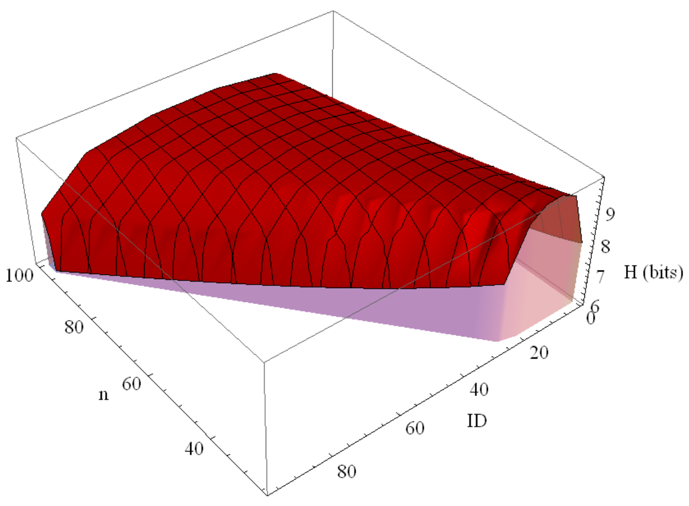

Fig. 10. Entropy variation for 11-bit IDs: effects of priority and number of IDs $n = 25..100$

mechanism requires even less than this since a dozen AES-CMAC computations are sufficient for securing a table of 100 IDs. This is good evidence that the proposed mechanism along with AUTOSAR recommendation would integrate well in real-world systems. We now discuss the expected security level for our mechanism.

Let the size of the ID space be σ where $\sigma \in \{2^{11}, 2^{29}\}$ and the number of IDs in the network be n. If order of the IDs must be preserved, then it is obvious that the i^{th} ID is not uniformly distributed in $[0..\sigma)$ because it is preceded by higher priority IDs and succeeded by lower-priority IDs. The probability of the i-th ID to have a value x can be computed as:

$$\Pr\Big[\mathsf{ID}_i = x | x \in [0..\sigma)\Big] = \binom{x-1}{i-1}\binom{\sigma-x}{n-i}\binom{\sigma}{n}^{-1} \quad (2)$$

Here $\binom{n}{k}$ denotes the binomial coefficient, i.e., $\frac{n!}{k!(n-k)!}$. In the above relation, $\binom{\sigma}{n}$ denotes the number of ways for picking n identifiers from the ID space of size σ. This term divides all occurrences with identifier i on position x, i.e., $\binom{x-1}{i-1}\binom{\sigma-x}{n-i}$, in order to obtain the probability that the i-th ID is on position x. Let $p_i(x)$ denote the previous probability, then the *entropy* of the ID (which defines its security level, i.e., the number of bits to be forged by an adversary) can be computed as:

$$H(\mathsf{ID}_i) = \sum_{x=i}^{\sigma-(n-i)} -p_i(x)\log_2 p_i(x) \quad (3)$$

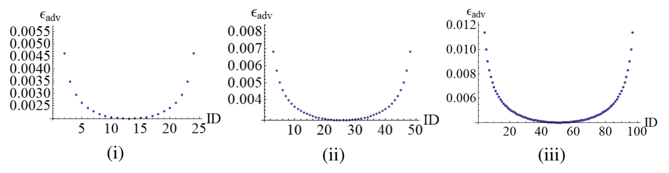

Fig. 11. Guessing probability of an adversary for each ID in case of 25, 50, 100 IDs

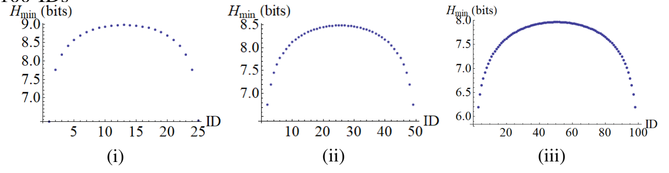

Fig. 12. Minimum entropy for each ID in case of 25, 50, 100 IDs

.

Note that summation starts from $x = i$ since at least $i-1$ values need to be allocated to smaller IDs and it stops at $\sigma - (n-i)$ since there are $n-i$ values for greater IDs. Figure 9 gives a graphical depiction on the variation of the ID entropy. In the right-most plot the variation of entropy in case of $n = 100$ IDs is shown. The entropy tops around 8.5 bits. Lower and higher priority IDs have a somewhat lower entropy at around 6 bits. Figure 10 translates the previous image to a 3D view in case of $n = 25..100$ IDs. Again, the lower and higher priority IDs have a slightly decreased entropy. The entropy increases if the number of IDs is lower, this is explainable as there is more room to allocate the IDs. We confirm these values by practical measurements over 10^6 extractions of the ID table.

The security level suggested by the entropy is not entirely correct with respect to adversary's capabilities. The best chances an adversary has is to guess the ID by using the value with the maximum probability from the distribution of the ID. But since the IDs are sorted, the distribution is non-uniform and some values have higher probability than others. In this respect, Shannon entropy, i.e., $H(\mathsf{ID}_i)$ in relation (3), is problematic since it sums over the entire distribution of the ID. This issue is well known in information security and to circumvent this issue, the *minimum entropy* is used as a metric, see for example [27], [9]. To define it, the *maximum guessing probability* of an adversary must be first defined as:

$$\epsilon_{adv}(\mathsf{ID}_i) = \mathrm{Max}\Big\{\Pr\big[\mathsf{ID}_i = x | x \in [0..\sigma)\big]\Big\} \quad (4)$$

Form which the *minimum entropy* follows as:

$$H_{\min} = \log_2 \frac{1}{\epsilon_{adv}(\mathsf{ID}_i)}, i = 1..n \quad (5)$$

Figure 11 shows the probability of success for an adversary in forging an ID and Figure 12 shows the minimum entropy of each ID. The plots are again for the case 25, 50 and 100 IDs.

For a crisper depiction of the guessing probability, min entropy and Shannon entropy, in Table I we provide values computed by simulation in case of 9 IDs on 11-bit and 29-bit out of $n = 100$ IDs. We selected the first three, last three and the three IDs in the middle to show that the distribution is non-uniform. For 11-bit IDs the values are computed based on

TABLE I
EXAMPLE OF GUESSING PROBABILITY, MIN ENTROPY AND SHANNON ENTROPY FOR SEVERAL IDS IN A DISTRIBUTION WITH $n = 100$ IDS

	ID no.	1	2	3	49	50	51	98	99	100
11-bit	ϵ_{adv}	4.6×10^{-2}	1.9×10^{-2}	1.3×10^{-2}	4×10^{-3}	4×10^{-3}	4×10^{-3}	1.4×10^{-2}	1.9×10^{-2}	4.8×10^{-2}
	$H_{\min}$	4.4	5.7	6.1	7.7	7.7	7.7	6.1	5.7	4.3
	H	5.7	6.6	6.9	8.7	8.7	8.7	6.9	6.6	5.7
29-bit	ϵ_{adv}	1.8×10^{-7}	6.8×10^{-8}	5.0×10^{-8}	1.4×10^{-8}	1.4×10^{-8}	1.4×10^{-8}	5.0×10^{-8}	6.8×10^{-8}	1.8×10^{-7}
	$H_{\min}$	22.3	23.7	24.2	25.9	26.0	26.0	24.2	23.7	22.3
	H	23.8	24.6	24.9	26.7	26.7	26.7	24.9	24.6	23.7

experimental data, for 29-bit IDs, since the space is too large, we computed the values synthetically based on approximations of the previous equations for guessing probability, min entropy and Shannon entropy. The variation compared to standard entropy is around 1 bit which suggests a security level that decreases by one bit. The situation is similar for 29 bit IDs, a case in which the security level is much higher (clearly, we recommend the use of extended identifiers for increasing the security level of the scheme).

III. EXPERIMENTS

In this section we first test the computational requirements of the proposed ID allocation algorithm then we test the protocol over a real-world ID allocation.

A. Experimental setup and computational results

The development boards that we use are an NXP S12XF512 Starter Kit and two Infineon Application Kits TC224 and TC297. The S12XF512 microcontroller can operate at a maximum frequency of 100MHz and has a memory of 512KB of Flash and 32KB of RAM respectively. The TC224 microcontroller operates at 133MHz and has 1 MB of Flash and 88 KB of RAM. The TC297 microcontroller has a maximum operating frequency of 300MHz and has 8 MB of Flash and 768 KB of RAM. As a measure of computational abilities, Table II illustrates the duration of computing MD5, HMAC-MD5 and CMAC-AES on each of the three employed platforms using a 64 bit input and a 128 bit key. These costs dozens microseconds on high-end cores and around 1-3 milliseconds on low end cores. In both cases they are affordable.

TABLE II
COMPUTATIONAL TIME FOR CRYPTO PRIMITIVES (MS)

Cryptographic function	Infineon TC224	Infineon TC297	Freescale S12XF
MD5	0.015	0.007	0.755
HMAC-MD5	0.030	0.015	3.126
CMAC-AES	0.055	0.016	1.383

On all controllers, we have measured the duration of generating the fresh ID table by CMAC-AES. The function was applied over 32-bit blocks concatenated to a 32-bit counter (the cycle identifier). We assumed a fixed secret key that is known by all nodes.

Table III shows computational results for building the ID table using binary sorted trees. Results are depicted both for

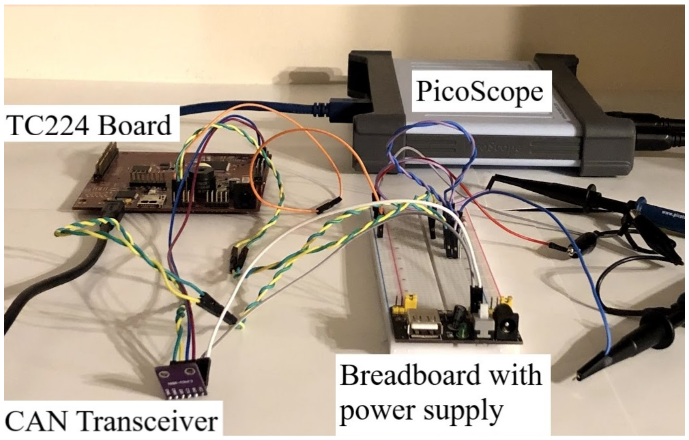

Fig. 13. Experimental setup with the TC224 board, external CAN Transceiver, breadboard and PicoScope for bus monitoring

the high-end Infineons where several hundred microseconds (or even less) are needed to update the ID table and for the low-end Freescale S12 where several milliseconds are needed. We depict both results for CMAC-AES for compatibility with existing standards, but also for a HMAC-MD5 which can be a faster instantiation. While MD5 is known to have collisions, this will not be of concern for this scheme since collisions cannot be computed in real-time as required for breaking the protocol. Moreover, we depict the case of a single MD5 computation which should be twice as fast as the HMAC-MD5. In this case we compute MD5 over the concatenation of the key with the message. This construction suffers from concatenation attacks, but again they are not feasible on the current scheme since the message has a fixed length. We keep the MD5 based instantiations as a bottom line for performance in terms of software implementation. Significant speedups can be achieved if hardware support is used. Such hardware already exists in many automotive-grade controllers and improvements should thus be within immediate reach. If the ID table is to be updated each $100ms$, the computations can be easily handled by both low-end and high-end cores. All cryptographic methods used in our implementation were integrated from the wolfSSL software library which is available on GitHub.

Table IV shows computational results for building the ID table using doubly linked lists. This approach has the advantage of removing recursion. Computational penalties exist but they are not high as the time to compute a full table of 100 IDs is kept at around $1ms$.

TABLE III
COMPUTATIONAL TIME FOR GENERATING THE ID TABLE USING BINARY SORTED TREES (MS)

		Infineon TC224			Infineon TC297			Freescale S12		
Cryptographic function	ID bits	25 IDs	50 IDs	100 IDs	25 IDs	50 IDs	100 IDs	25 IDs	50 IDs	100 IDs
MD5	11	0.100	0.157	0.295	0.086	0.140	0.271	3.576	6.468	13.410
MD5	29	0.151	0.280	0.524	0.120	0.222	0.422	5.100	11.330	26.320
HMAC-MD5	11	0.157	0.243	0.460	0.112	0.178	0.346	9.872	23.100	46.660
HMAC-MD5	29	0.251	0.464	0.863	0.164	0.302	0.567	22.570	45.420	95.120
CMAC-AES	11	0.260	0.397	0.678	0.129	0.198	0.347	4.618	10.870	23.780
CMAC-AES	29	0.432	0.802	1.487	0.186	0.334	0.618	10.360	20.970	44.480

TABLE IV
COMPUTATIONAL TIME FOR GENERATING THE ID TABLE BY DOUBLY LINKED LISTS (MS)

		Infineon TC224		
Cryptographic function	ID bits	25 IDs	50 IDs	100 IDs
MD5	11	0.170	0.296	0.542
MD5	29	0.200	0.320	0.674
HMAC-MD5	11	0.271	0.424	0.737
HMAC-MD5	29	0.300	0.478	1.000
CMAC-AES	11	0.421	0.591	1.003
CMAC-AES	29	0.480	0.810	1.600

B. Destroying frames with malicious IDs

We have also put to test the idea of destroying malicious frames. This concept has been previously explored in [20] and [17]. In order to read the ID bits from each CAN message (before actually receiving the whole message) we monitor the TC224 RX pin which is connected to the RX pin of the external CAN transceiver as shown in Figure 13. Before reading the message ID, we perform a loop which waits for the start bit to be sent on the bus (a dominant bit). After the start bit, we read 14 RX-pin states, with $2\mu s$ delay between each read out, corresponding to the bit states on the bus when using a 500 kbps bit rate. Based on the bits read, we rebuild the 11-bit ID by filtering out stuffing bits (if they are present). Having the fresh ID table, we use the binary sorted tree to search for each ID extracted from the bus. If the message identifier is not found in the ID reallocation tree, the bus is forced to a dominant level using the TC224 TX pin which is connected to the external CAN transceiver TX pin.

The time for checking the ID in the table is very fast, it varies from $0.162\mu s$ to $0.850\mu s$ with an average of $0.555\mu s$ (in case of 90 IDs in the table). After the check, we wait for the end of frame bits on the bus (7 consecutive recessive bits). Next, we return to the first loop again and wait for the start bit of a new CAN message. A plot with a frame being destroyed on the bus (after a failed ID check) is shown in Figure 14. Note that the first two genuine frames are successfully sent on the bus.

C. Bus results from traffic logs

The bus traffic allocation that we use corresponds to a trace with 88 IDs that we recorded in a high-end vehicle. The bus speed was set at 500 kbps. This is a high number of IDs, usually nodes are placed in subnetworks that have several dozens of IDs. We endeavored to test our ID allocation mechanism in this case as representative for a worst-case scenario. The original IDs are on 11 bits, we kept the same cycle time for them when experimenting with the trace. When broadcasting traffic with newly allocated IDs, we kept an inter-frame space (IFS) of at least $250\mu s$. During the IFS the nodes can compute all or a part of the new ID table. Generally, in our implementation the ID table was updated with 5–10 new IDs during the IFS, until all the 88 IDs were filled. The fresh ID table was always ready at 100ms, 200ms, 250ms, 500ms or 1s (depending on the experiment). In order to verify the messages transmitted on the bus for each experiment and to log all the information we used a PicoScope 5000 Series connected to the CAN differential lines. Bus data was analyzed with the oscilloscope software which is available for free. This allowed us to check the interpretation of each CAN frame and store the log for offline analysis.

Figure 15 shows the ID allocation for the first 3 seconds of runtime. First, the original IDs are shown in (i), the pattern in which they appear is obvious and they can be easily forged by an adversary. Then we show the IDs re-allocated by ordered CMACs in the 2^{11} address space with $\Delta = 100ms$ update in (ii) and $\Delta = 500ms$ update in (iii). On the 11-bit identifier map (ii) the IDs are somewhat clustered at the bottom of the figure (this is because of the reduced space for allocating the IDs). We move to the 2^{29} address space in (iv) and (v), with updates each $500ms$ and $100ms$ respectively. For the 29-bit map at $100ms$ the distribution is very close to the random allocation in (vi) which we use only for comparison as priority is not preserved by it. Note that while the six pictures look entirely distinct, the CAN bus traffic is actually identical in all pictures, i.e., (ii)–(v) all reduce to (i) after re-mapping the IDs with the correct identifier table (this operation can be easily performed by all genuine nodes that are in possession of the table). As stated, the security level is around 6–9 bits for standard identifiers and 24-27 bits for extended identifiers. For obvious security reasons, we recommend extended 29-bit identifiers which significantly increase the security level.

ID filtering issues and solutions. The proposed mechanism may impede regular ID filtering that is performed by CAN controllers. This mechanism was designed on CAN such that nodes will not be disturbed by frames that are out of scope for their tasks. This problem may be easily circumvented by performing filtering at the software layer after the IDs are mapped to their original values. This would only require some bit operations on the IDs (a logical AND may be sufficient) that is fast to perform and should not cause concerns. If ID filtering must be preserved at the hardware layer, then at least part of the ID bits should be left unchanged to be used for

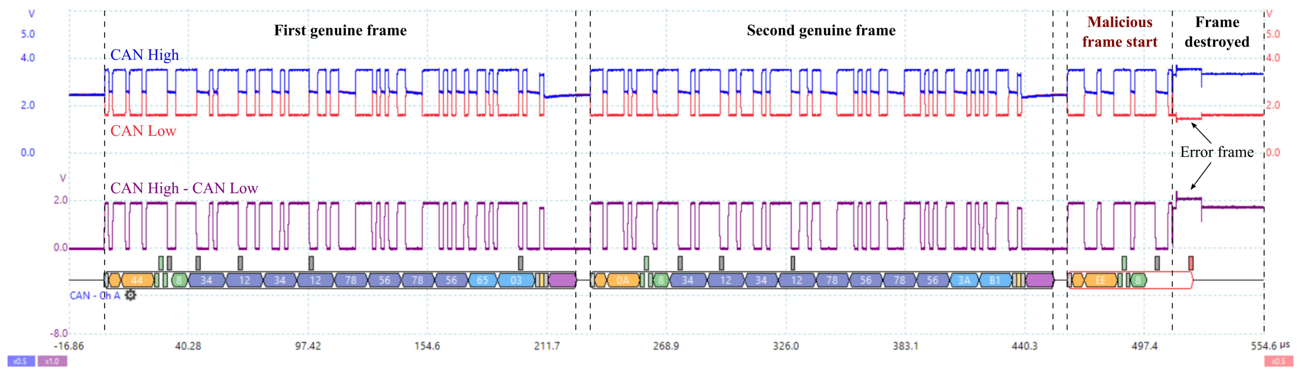

Fig. 14. Two genuine frames followed by an unrecognized ID that is destroyed by dominant bits (PicoScope plot)

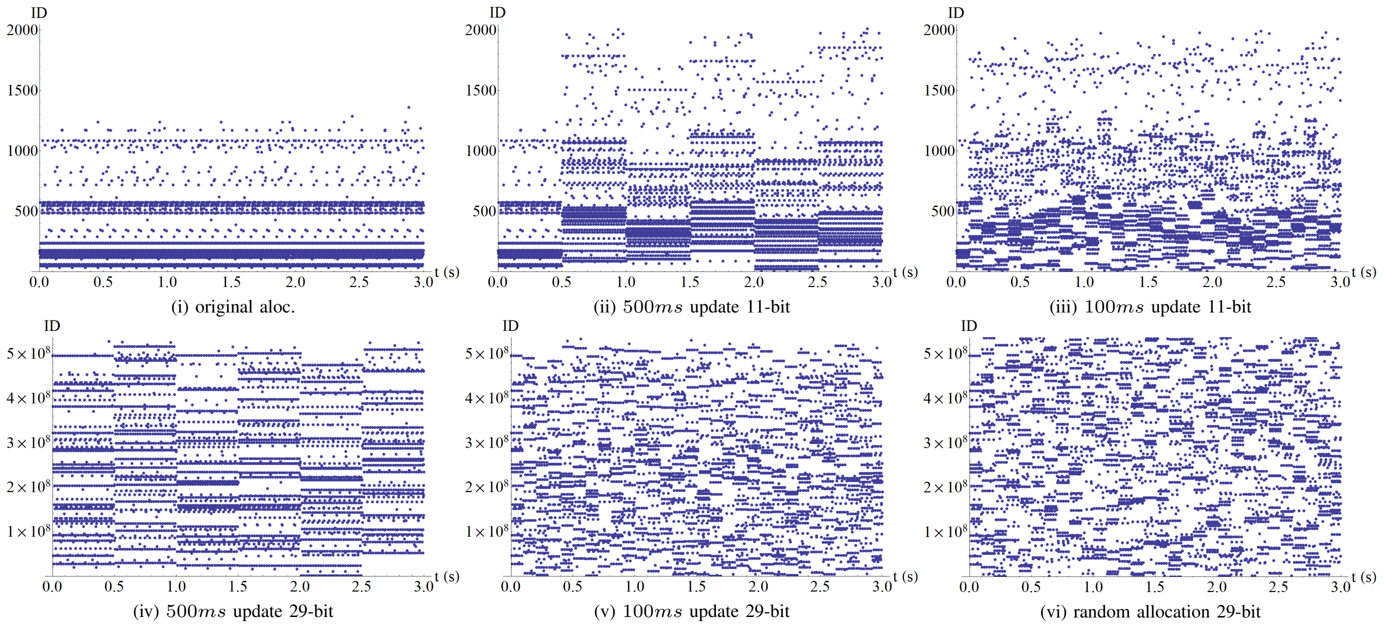

Fig. 15. IDs arriving on the bus during the first 3 seconds of runtime: (i) original ID allocation, (ii) re-allocated by ordered CMACs on 11-bit with $\Delta = 500ms$ update and (iii) re-allocated by ordered CMACs 11-bit at $\Delta = 100ms$ update, (iv) re-allocated by ordered CMACs 29-bit at $\Delta = 500ms$ update, (v) re-allocated by ordered CMACs 29-bit at $\Delta = 100ms$ update, and (vi) random allocation (identical traffic in all six pictures)

filtering. This would however lower the security level.

D. Biasing the ID distribution

A potential shortcoming of the ID distribution is that the highest priority IDs get the lowest entropy and thus the lowest security level. Similarly, low priority IDs will get low entropy but this may be less important as they likely carry less significant messages. The entropy is the highest for IDs in the middle of the distribution.

If low entropy for high priority IDs causes concerns, then this can be fixed by at least two approaches. One straightforward solution is to simply map high priority IDs that need more security to values in the middle of the distribution (middle-valued ID get the highest entropy). This will decrease their priority however, and if this is undesired, another approach is to bias the distribution of the IDs.

In principle, we want to change the cumulative distribution function (CDF) of the ID distribution. One way to do this is to flip a bit b and if $b = 0$ leave the newly generated ID unchanged, else, if $b = 1$, simply set the most significant bit (MSB) of the ID to 1. Let $\Pr[b = 1] = \xi$ then approximately ξ percent of the IDs will be shifted above 2^{10}, which leaves more room for high priority IDs (located in the lower part of the distribution). The computational time will slightly increase since there will be slightly more collisions and more fresh values for the IDs need to be generated. The probability that the *i-th* ID is not biased can be computed by summing over the probability mass function of n independent Bernoulli trials with success probability ξ:

$$\Pr\left[\neg biased\right] = \sum_{j=i}^{n} \binom{n}{j} \xi^j (1-\xi)^{n-j} \quad (6)$$

This requires that at least i of the IDs where not biased. As can be seen in part (i) of Figure 18 (which depicts the probability of an ID not being biased for a $\xi = 0.25$, $\xi = 0.5$, $\xi = 0.75$), the probability cuts sharply around the expected value ξn.

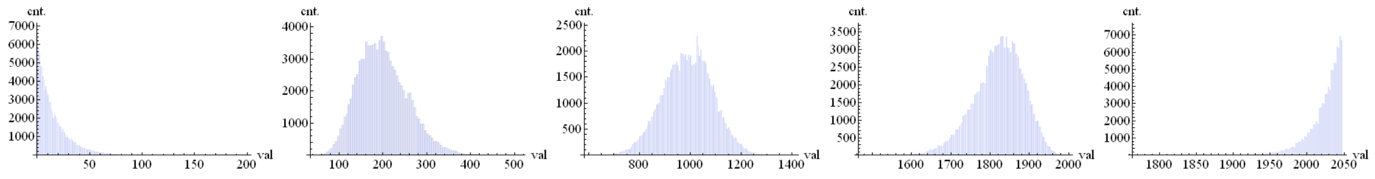

Fig. 16. Histogram distribution for $\mathsf{ID}_1, \mathsf{ID}_{10}, \mathsf{ID}_{50}, \mathsf{ID}_{90}$ and ID_{100}

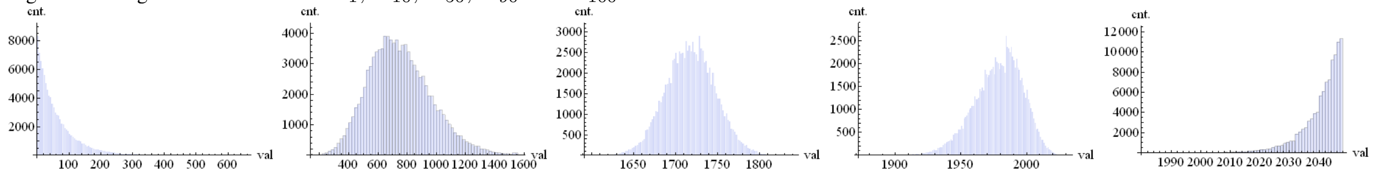

Fig. 17. Histogram distribution with biased selection for $\mathsf{ID}_1, \mathsf{ID}_{10}, \mathsf{ID}_{50}, \mathsf{ID}_{90}$ and ID_{100}

For this reason, we can get a rough approximation of the entropy by simply considering that for all IDs smaller than ξn, their entropies are upper bounded by the entropy of ξn IDs distributed in a space of size σ (this is defined in relation (2) and the same holds for relation (3), (4)). For IDs that are higher than ξn, their entropies are lower bounded by the entropy of n IDs distributed in the $\sigma/2^\ell$ space, where ℓ denotes the bits in the mask. In parts (ii), (iii) and (iv) of Figure 18 we show the effects of biasing on entropy, guessing probability and minimum entropy of each ID. These values are only rough estimations based on the previous reasoning, but with a bias $\xi = 0.25$ and a bit mask of $\ell = 1$ bit the entropy of higher priority IDs is roughly increased by 1-bit. Various distributions can be tested, our intention here was just to give a brief overview of this procedure.

For a better understanding on the variation of guessing probability, min entropy and Shannon entropy we now discuss simulation results obtained for 100 IDs in case of the uniform and biased distributions. Figure 16 shows the histogram distribution for the first ID, middle ID (50) and last ID out of 100 IDs (in descending order of priority) in case of the regular distribution from the previous section. While the distribution of the middle ID is Gaussian for the first and last it is biased to the left and right side respectively (the entropy that we determined experimentally is close to the theoretical one, i.e., 6-9 bits). When extending the identifier space to 29-bits, the same computations yield around 24–27 bits of entropy which is quite high. We also show the results for the biased distribution in Figure 17. It can be easily seen that in contrast to Figure 16, all the IDs are shifted to the right. The distribution of ID_1 and ID_{10} is wider (for ID_1 it is 4 times wider as it expands to 200 rather than 50 in the normal case). In contrast, the distribution of ID_{50}, ID_{90} and ID_{100} is now narrower and shifted to the right. The plots in the introductory section, i.e., Figures 2 and 3, give a clear overview of what happens in the ID allocation during 3 seconds of runtime. Based on experimental results, by using the previously defined relations, we determined that the entropy of the first ID increases with roughly 2 bits.

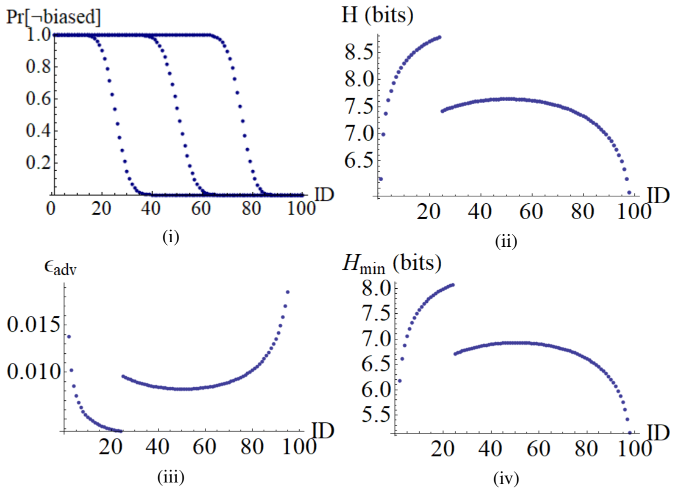

Fig. 18. Probability of an ID not being biased ($\xi = 0.25$, $\xi = 0.5$, $\xi = 0.75$) and rough estimation of entropy (ii), adversary advantage (iii) and minimum entropy (iv) in case of a biased distribution ($\xi = 0.25$) for 100 IDs

IV. Conclusion

We provide a highly efficient software mechanism for authenticating sender nodes on the CAN bus. The mechanism requires several hundred micro-seconds to generate a fresh ID table and IDs that are not genuine can be verified in less than a micro-second and destroyed by dominant bits (all these are done from the software layer). By garnering 8-9 bits of entropy from 11-bit identifiers and 24-27 bits from extended 29-bit identifiers and still preserving the order of the IDs on the bus we are close to the maximum security level that can be achieved by using the identifier field. A biased distribution of the IDs can be used in order to increase the entropy of IDs that carry more important messages. Our mechanism can easily complement security specifications in existing standards and increase the security level which is quite weak due to the limited payload of CAN frames. The computational requirement are almost insignificant, several CMAC-AES being sufficient to generate a large ID table, e.g., less than 10 computations are needed for a table of 100 identifiers. The life-time of such an identifier table depends on specific security needs, e.g., from dozens of milliseconds to seconds. Our experimental results show that the procedure is feasible both on low-end and high-end microcontrollers. The mechanism does not impede existing proposals for intrusion-detection systems based on frame IDs since the randomized IDs can be easily re-mapped to the original values.

Acknowledgement. This work was supported by a grant of Ministry of Research and Inovation, CNCS-UEFISCDI, project number PN-III-P1-1.1-TE-2016-1317, within PNCDI III (2018-2020).

References

[1] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu. Order preserving encryption for numeric data. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 563–574. ACM, 2004.

[2] AUTOSAR. *Specification of Secure Onboard Communication*, 4.3.1 edition, 2017.

[3] G. Bella, P. Biondi, G. Costantino, and I. Matteucci. Toucan: A protocol to secure controller area network. In *Proceedings of the ACM Workshop on Automotive Cybersecurity*, pages 3–8. ACM, 2019.

[4] A. Boldyreva, N. Chenette, and A. O'Neill. Order-preserving encryption revisited: Improved security analysis and alternative solutions. In *Annual Cryptology Conference*, pages 578–595. Springer, 2011.

[5] 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.

[6] K.-T. Cho and K. G. Shin. Fingerprinting electronic control units for vehicle intrusion detection. In *25th USENIX Security Symposium*, 2016.

[7] K.-T. Cho and K. G. Shin. Viden: Attacker identification on in-vehicle networks. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1109–1123. ACM, 2017.

[8] 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*, 2018.

[9] B. Espinoza and G. Smith. Min-entropy as a resource. *Information and Computation*, 226:57–75, 2013.

[10] B. Groza and P.-S. Murvay. Security solutions for the controller area network: Bringing authentication to in-vehicle networks. *IEEE Vehicular Technology Magazine*, 13(1):40–47, 2018.

[11] B. Groza, L. Popa, and S. Murvay. INCANTA - intrusion detection in controller area networks with time-covert cryptographic authentication. In *International Workshop on Cyber Security for Intelligent Transportation Systems (ESORICS'18 Workshops)*, 2018.

[12] K. Han, A. Weimerskirch, and K. G. Shin. A practical solution to achieve real-time performance in the automotive network by randomizing frame identifier. In *Proc. Eur. Embedded Secur. Cars (ESCAR)*, pages 13–29, 2015.

[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] 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*, pages 19–26. ACM, 2017.

[15] ISO 15765-2:2016 Road vehicles — Diagnostic communication over Controller Area Network (DoCAN) — Part 2: Transport protocol and network layer services. Standard, ISO, Apr. 2016.

[16] M. Kneib and C. Huth. Scission: Signal characteristic-based sender identification and intrusion detection in automotive networks. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 787–800. ACM, 2018.

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

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

[19] C.-W. Lin, Q. Zhu, and A. Sangiovanni-Vincentelli. Security-aware modeling and efficient mapping for CAN-based real-time distributed automotive systems. *IEEE Embedded Systems Letters*, 7(1):11–14, 2015.

[20] T. Matsumoto, M. Hata, M. Tanabe, K. Yoshioka, and K. Oishi. A method of preventing unauthorized data transmission in controller area network. In *Vehicular Technology Conference (VTC Spring), 2012 IEEE 75th*, pages 1–5. IEEE, 2012.

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

[22] M. R. Moore, R. A. Bridges, F. L. Combs, M. S. Starr, and S. J. Prowell. Modeling inter-signal arrival times for accurate detection of can bus signal injection attacks: a data-driven approach to in-vehicle intrusion detection. In *Proceedings of the 12th Annual Conference on Cyber and Information Security Research*, page 11. ACM, 2017.

[23] J. Ning, J. Wang, J. Liu, and N. Kato. Attacker identification and intrusion detection for in-vehicle networks. *IEEE Communications Letters*, 2019.

[24] ODVA. *The CIP Networks Library Vol. 3: DeviceNet Adaptation of CIP, Edition 1.10*, November 2010.

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

[26] S. U. Sagong, X. Ying, A. Clark, L. Bushnell, and R. Poovendran. Cloaking the clock: emulating clock skew in controller area networks. In *Proceedings of the 9th ACM/IEEE International Conference on Cyber-Physical Systems*, pages 32–42. IEEE Press, 2018.

[27] G. Smith. On the foundations of quantitative information flow. In *International Conference on Foundations of Software Science and Computational Structures*, pages 288–302. Springer, 2009.

[28] 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*, pages 63–68. IEEE, 2016.

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

[30] S. Woo, H. J. Jo, I. S. Kim, and D. H. Lee. A Practical Security Architecture for In-Vehicle CAN-FD. *IEEE Trans. Intell. Transp. Syst.*, 17(8):2248–2261, Aug 2016.

[31] 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*, 7:15521–15536, 2019.

[32] 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*, 6:54607–54623, 2018.

[33] Y. Xie, G. Zeng, R. Kurachi, H. Takada, and G. Xie. Security/timing-aware design space exploration of can fd for automotive cyber-physical systems. *IEEE Transactions on Industrial Informatics*, 15(2):1094–1104, 2018.

[34] 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*, 7:95418–95429, 2019.

[35] X. Ying, G. Bernieri, M. Conti, and R. Poovendran. Tacan: Transmitter authentication through covert channels in controller area networks. In *Proceedings of the 10th ACM/IEEE International Conference on Cyber-Physical Systems*, pages 23–34. ACM, 2019.

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 (2018-2020), two research programs dedicated to automotive security funded by the Romanian National Authority for Scientific Research and Innovation.

Lucian Popa started his PhD studies in 2018 at Politehnica University of Timisoara (UPT). He graduated his B.Sc in 2015 and his M.Sc studies in 2017 at the same university. He has a background of 4 years as a software developer and later system engineer in the automotive industry as former employee of Autoliv (2014 - 2018) and current employee of Veoneer (2018 - present). His research interests are in automotive security with focus on the security of in-vehicle buses.

Pal-Stefan Murvay is 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.