This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2018.2869351, IEEE Transactions on Information Forensics and Security

Efficient Intrusion Detection with Bloom Filtering in Controller Area Networks (CAN)

Bogdan Groza and Pal-Stefan Murvay

***Abstract*—Due to its cost efficiency the Controller Area Network (CAN) is still the most wide-spread in-vehicle bus and the numerous reported attacks demonstrate the urgency in designing new security solutions for CAN. In this work we propose an intrusion detection mechanism that takes advantage of Bloom filtering to test frame periodicity based on message identifiers and parts of the data-field which facilitates detection of potential replay or modification attacks. This proves to be an effective approach since most of the traffic from in-vehicle buses is cyclic in nature and the format of the data-field is fixed due to rigid signal allocation. Bloom filters provide an efficient time-memory tradeoff which is beneficial for the constrained resources of automotive grade controllers. We test the correctness of our approach and obtain good results on an industry-standard CANoe based simulation for a J1939 commercial-vehicle bus and also on CAN-FD traces obtained from a real-world high-end vehicle. The proposed filtering mechanism is straight-forward to adapt for any other time-triggered in-vehicle bus, e.g., FlexRay, since it is built on time-driven characteristics.**

I. Introduction and motivation

Recently introduced communication interfaces such as FlexRay or BroadR-Reach incur higher production costs which are poorly justified for low or mid-end vehicles that form a relevant segment of the market. In contrast, the CAN bus is a cost-efficient solution and its reliability is proved by at least three decades of use. However, when it comes to security all in-vehicle buses are lacking.

Designed by Bosch in 1983, the CAN bus is a differential two wire bus with arbitration based on the priority of the message identifier (the lower the ID, the higher the message priority). A typical network topology is suggested in Figure 1 where several nodes are placed inside a car on the two wire CAN bus. Real-world network topologies may be complex, with dozens of ECUs linked to one sub-network and several sub-networks inside a single car, but mutatis mutandis the same bus-oriented topology stays at the core of the network. The structure of the CAN frame is suggested in Figure 2. The frame encloses a data-field of at most 64 bits and the identifier field (ID) which has 11 bits in standard frames and 29 bits in extended frames. Recently, BOSCH proposed an extension of the three decades old CAN with the newer CAN-FD (CAN with Flexible Data-Rate) that carries up to 64 bytes of data (instead of 64 bits). This proves serious intentions in keeping the CAN bus on the market despite the existence of newer alternatives.

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

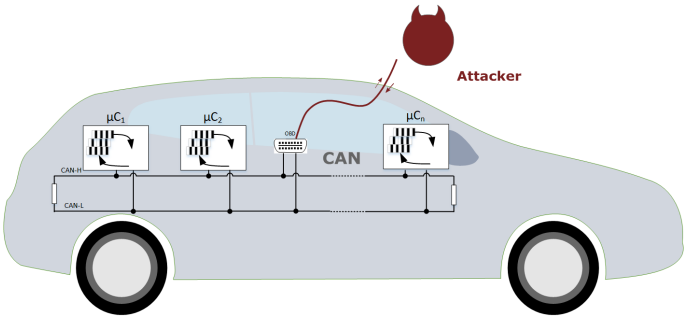

Fig. 1. Application setting: an adversary intercepts and injects frames on the CAN bus, nodes are filtering the received packets

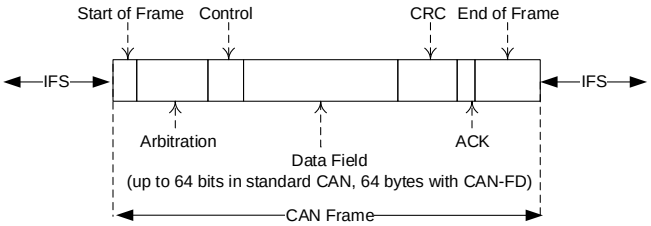

Fig. 2. Structure of a CAN frame

By default, the CAN bus, and similarly CAN-FD, has no intrinsic security mechanisms, except for standard CRC codes. Consequently, recent attacks proved that an adversary that has access to the CAN bus can take full control over the car by locking the brakes, controlling the steering column, etc. The seminal works in [19], [3] opened the road for a significant number of attacks that were reported in the past few years. A comprehensive practical analysis is available in [29].

In response to these, a number of security protocols for the CAN bus were proposed. Some proposals rely on regular message authentication codes (MACs) and symmetrically shared secret keys, e.g., [13], [44], [46] and [36]. Other proposals extend these concepts with efficient signal allocation to accommodate both regular signals and cryptographic MACs [25], [26]. TESLA-like protocols which were highly effective in sensor networks are discussed for the CAN bus in [12]. Group key-sharing between nodes is proposed in [11]. Other works use physical properties of the bus in order to secretly exchange cryptographic keys [16], [31] or to recognize nodes based on physical characteristics of the signal [32]. Additional hardware is introduced by [20] to discard attacker messages by error flags.

Contribution in brief. Our main motivation stems from

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2018.2869351, IEEE Transactions on Information Forensics and Security

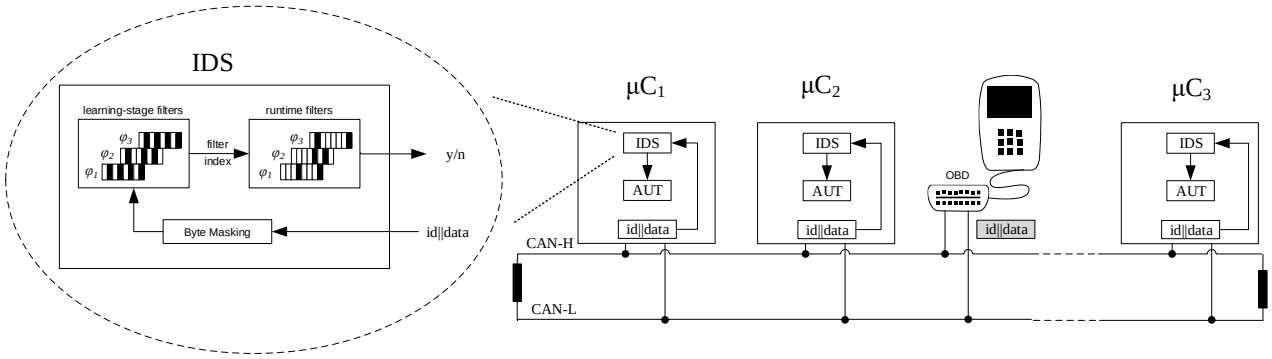

Fig. 3. Topology of the CAN bus and node setup, suggestive depiction of the proposed IDS to the left

the fact that the large majority of practical attacks, e.g., the attacks from [19], [29], are done by replaying frames that were previously recorded on the bus or by injecting modified versions of already recorded frames. These are clear violations of frame periodicity and/or structure. Since communication on the CAN bus is done mostly within precise communication cycles and signal allocation within a specific frame is rigid, it appears feasible to detect such intrusions. Consequently, in this work we design and evaluate an intrusion detection mechanism based on the frame periodicity and its content by accounting for parts of the data-field that do not change often.

Since it is inconvenient to store all the required information for each frame (due to obvious memory constraints), we use Bloom filters to identify messages along with their content provided that they are broadcast at fixed time intervals. The size of the IDs, i.e., 11 bits for regular frames and 29 bits for extended frames, along with an 8 to 64 byte data-field makes it inefficient to store all this information for each frame on each node. In contrast, Bloom filters [2] allow a time-memory trade-off for testing set membership which makes them suitable for a large number of networking applications. Briefly, we store only several Bloom filters and a byte-mask for each ID which requires a single bit for each byte of the message. By using these we filter-out the most significant amount of frames, which are the cyclic frames for periodic reports. On-event frames can be treated by distinct, less efficient mechanisms, without causing much disturbances on the bus since they form a smaller amount of the total traffic. Our proposal addresses CAN in general, regardless of its actual embodiment, be it the older CAN or the newer CAN-FD. In the experimental results we cover datasets from both these implementations.

Details of our application setup are graphically depicted in Figure 3 which presents several nodes linked to the CAN bus. An external device, potentially malicious, is also linked to the bus based on the On-board diagnostics (OBD) port. The external device sends messages on the bus which are represented by an ID and data-field. Each node receives the messages, passes them through the Intrusion Detection System (IDS) and then through some authentication mechanism that may be in place but which is beyond the scope of this work. The IDS is expanded to the left of the figure. As detailed on the left side of the figure, the received frames first go to the byte-masking mechanism, then they are verified by the learning-stage filters and by the runtime filters in order to decide if an intrusion took place.

A. Related work on intrusion detection

While intrusion detection systems (IDS) have a long history in computer networks, the use of IDS for in-vehicle buses is a more recent topic. One of the earliest discussions for intrusion detection on the CAN bus can be found in [14]. This solution accounts for the number of occurrences of each message in a specific time-slot, thus detecting potential replay or injection attacks. An extended discussion on security threats and counter-measures, including the use IDS can be found in [15].

In the past few years, the number of IDS proposals for CAN has surged. The use of message periodicity seem to frequently re-occur in designing intrusion detection on CAN, e.g., [30] and [38]. Neural networks were proposed to detect intrusions on the CAN bus in [17], [18] and [41]. However, the test-bed from [17] is based on the OCTANE automotive network simulator [9] and unfortunately contains only three CAN ids (a very small number) which makes it unclear if the content of the data-field is representative for genuine in-vehicle traffic. Hidden Markov models for detecting anomalies on in-vehicle traffic were employed in [35]. Other techniques that were recently explored include: machine learning [43], multivariate time series [42] and regression learning [23]. A more formal approach based on finite-state automatons is considered in [40]. In [28] the authors proposed that ECUs that see messages on the bus with their own ID should discard the message with an error flag before transmission is completed. This proposal can not prevent the case when the genuine ECU is unplugged from the network or is in bus off.

The use of anomaly detection sensors is discussed in [34]. Entropy is considered in [33] and [27]. Hardware implementations based on the error-confinement mechanism of CAN are discussed in [10]. Other lines of work are focused on physical characteristics, e.g., clock-skews [5] or voltage levels [32], [7], [6] to detect anomalies on the bus. The delay between regular frames and remote frames is used by the authors in [21] to decide if intrusions occur on the bus. However, recently the clock-skew based detection mechanism in [5] was proved vulnerable to cloaking attacks [37] in which a malicious node modifies the timing of the frames he sends to mimic a genuine

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2018.2869351, IEEE Transactions on Information Forensics and Security

node from the bus. This proves that intrusion detection systems based on timing alone, such as the ones in [5], [21], may be quite fragile. In contrast to these, besides accounting for the timing of the frame our proposal addresses the content of the data-field.

Analyzing the behaviour of in-vehicle ECUs, i.e., detecting anomalies, is put in a distinct context in [45] where anomalies are used to detect manipulated in-vehicle ECUs. Techniques for intrusion-detection on the CAN bus are far reaching as recent work discusses applications to an avionics bus [39].

As the previous enumeration points out, proposals from related work are hard to compare as they use very distinct mechanisms. What makes such a comparison even harder is that all approaches use distinct datasets making it impossible to confront the success rate of the intrusion detection algorithm. For example [27] uses data from a Ford Fiesta and [5] from Honda Accord, Dodge RAM and a Toyota Camry. What is relevant however, is that most of the relevant proposals validate their effectiveness on real-world data. In our work we use the output of an industry-standard CANoe simulation and a CAN-FD trace from a real-world high-end vehicle.

In contrast to related work, we consider that the advantage of Bloom filtering is that it sets way for a memory-efficient analysis of various parts of the data field. Extensive results on Bloom filtering exist and these may benefit from a new application area. Besides Bloom filters, our solutions exploits the periodicity of messages and their entropy which is also validate by related work. As a tool in our analysis, we also use Hamming distances which have been recently suggested for IDS on CAN in [8].

II. Quantitative analysis of CAN bus traffic

For a crisper image over the data on which we apply Bloom filtering, we first give a quantitative analysis of CAN bus data.

A. Tools for analysis and results

Our analysis is focused on the number of IDs, their periodicity and and the entropy carried by the data-field associated to a particular ID. As shown by the analysis that follows, a significant number of bits from the data-field have fixed values. Fixed bits are decreasing the entropy of the frame and can be efficiently exploited in filtering. We emphasize that while our analysis is purely quantitative, automotive manufacturers have additional knowledge on the particular values of these bits and even more efficient filters can be design based on procedures that are similar to ours. For brevity, we first formalize the notions that we use for analyzing the CAN trace.

We denote the trace $\mathcal{T}$ recorded on the CAN bus as the collection of identifier-message pairs (id_i, m_i) of length ℓ, i.e.,

$$\mathcal{T} = \{(id_1, m_1), (id_2, m_2), ..., (id_\ell, m_\ell)\}$$

We consider that the ID-message pairs from the trace occur in the order that they are recorded on the bus, thus identical pairs may repeat. For the analysis in this section the arrival time of the frame is of no importance but it will be used in the filtering procedure. Also, we are not intersted in particular violations of the CAN protocol, e.g., wrong CRCs, malformed frames, etc., which are addressed by the CAN error management layer.

Let $\mathcal{T}_{id'} = \{(id_i, m_i) | id_i = id', i = 1..\ell'\}$ be the trace containing identifier-message pairs only for identifier id'. As a straight-forward measurement for variations of the data-field we use the tuple of *intra-distances* associated to a particular ID which is defined as the Hamming distance between the bit representation of each two messages from the trace $\mathcal{T}_{id'}$, i.e.,:

$$\mathcal{I}_{id'} = \text{Hamming}(m_i, m_j), \forall i, j \in [1..\ell']$$

We assume that for each identifier id' the length of the message, i.e., the data-field, is fixed to a value $\delta(id')$. This is consistent to the results of our experiments and even if this is not the case our framework can be easily adapted for variable length messages.

The entropy carried by each frame is a relevant indicator for the amount of bits that stay constant and can be used for filtering. For assessing the entropy carried by a message associated to a particular ID we first measure the guessing probability of each byte of the message and then we define the minimum entropy based on this probability (this is a standard procedure in defining the minimum entropy of a random variable). Let $\mathrm{B}_j^{id'}, j = 1..\delta(id')$ be the bytes of the data-field associated to identifier id', the data-field bytes define a random variable over the trace $\mathcal{T}_{id'}$ (the trace is a set of ℓ' trials for each byte). The *guessing probability* of each byte from m_i is defined as the maximum probability that the byte takes a particular value, i.e.,

$$\gamma_{\mathrm{B}_j^{id'}} = \max\left\{\Pr[\mathrm{B}_j^{id'} = v] : \ v \in \{0..255\}\right\}, \forall j = 1..\delta(id').$$

By definition, the *minimum entropy* of the message byte m_j is $\log_2(1/\gamma_{\mathrm{B}_j^{id'}})$ and consequently we can define *the minimum entropy* of the data-filed for identifier id' as the sum of entropies for all the corresponding message bytes, i.e.,

$$\mathrm{H}_{id'} = \sum_{i=1,\delta(id')} \log_2(1/\gamma_{\mathrm{B}_j^{id'}}).$$

We also define the *byte-mask* of identifier id' as a tuple of zeros and ones that show whether a particular byte is constant or not, i.e., $\mathbb{M}_{id'} = \mathrm{b}_0, \mathrm{b}_1, ..., \mathrm{b}_{\delta(id')}$ where $\mathrm{b}_j = 1 \Leftrightarrow \gamma_{\mathrm{B}_j^{id'}} = 1$ and $\mathrm{b}_j = 0 \Leftrightarrow \gamma_{\mathrm{B}_j^{id'}} \neq 1$ (note that constant bytes by definition have 0 entropy and equivallently their guessing probability equals 1).

B. Results

We first analyze the traces from a J1939 CANoe simulation. For generating the SAE J1939 traffic we employed a sample CANoe configuration that comes with the installation of this environment. The configuration simulates a network with two sub-buses connected by a gateway node. One sub-network has an additional 5 nodes while only one extra node is present on the other.

In Figure 4 we depict the content of the frame as recorded over the first 32 instances of a specific frames, the we depict

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2018.2869351, IEEE Transactions on Information Forensics and Security

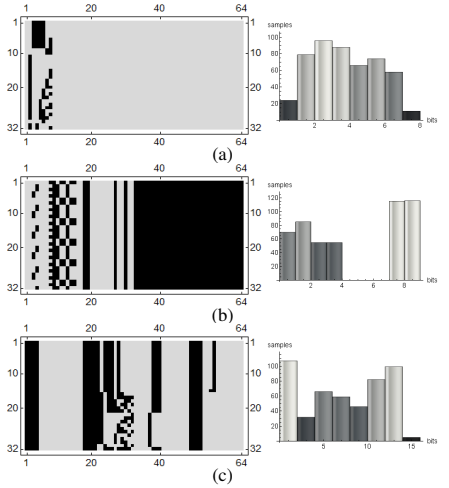

Fig. 4. Graphical depiction of frame content (left) and histogram of hamming inter-distances (right) for three CAN frames from J1939 CANoe simulation: 4-bit entropy (a), (b) and 13-bit entropy (c)

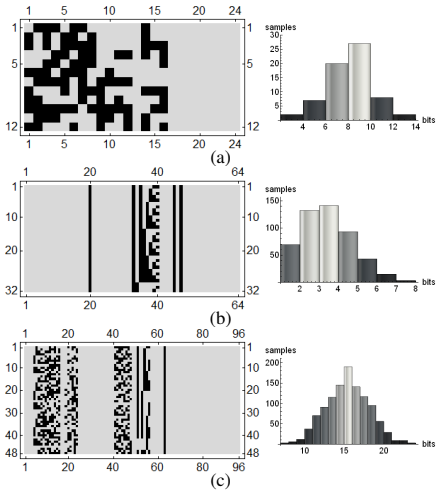

Fig. 5. Graphical depiction of frame content (left) and histogram of hamming inter-distances (right) for: smallest 24-bit frame with 13-bit entropy (a), regular 64-bit CAN frame with 12-bit entropy (b), and a 96-bit frame with 30-bit entropy (c)

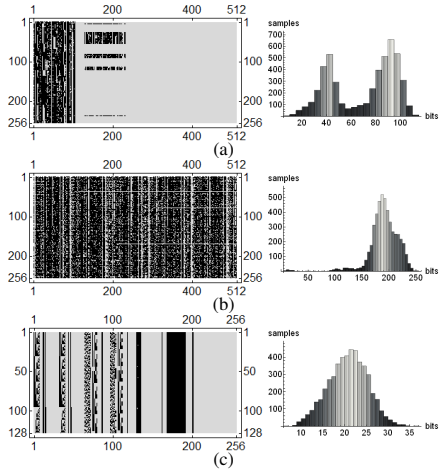

Fig. 6. Graphical depiction of frame content (left) and histogram of hamming inter-distances (right) for: large 512-bit frame with 61-bit entropy (a), large 512-bit frame with 203-bit entropy (b), average 256-bit frame with 60-bit entropy (c)

the Hamming inter-distances as histogram distributions as computed over all frames with the same ID from the trace. We first show two frames with low entropy of 4 bits (a, b) and then a frame that has higher entropy of 13 bits (c). This represents the maximum entropy that can be extracted from frames in the CANoe J1939 simulation, generally, frames had a very small amount of entropy. The trace that we analyzed had around 25,000 frames.

We proceed to the analysis of the real-world CAN trace in Figure 5, the trace contained almost 1 million frames. We first depict the results for frames of 24–96 bits. We display the content of the frame as recorded over the first few dozen packets along with the Hamming inter-distances as histogram distributions in Figure 5 (a-c) as computed over all frames with the same ID in the trace. Figure 6 (a-c) gives similar depictions for large frames of 256 and 512 bits. The entropy of real-world frames is much higher and the Hamming inter-distances form a clear Gaussian distribution. This is not only due to the larger data-set as it can be clearly seen that even for the first few frames bit variations are significantly higher than in the case of the CANoe simulation.

In Figure 7 we give a graphical depiction of the entropy and the length of the byte-masks for smaller frames of 24 to 160 bits. Figure 8 does the same for larger frames of 192 to 512 bits. As shown in the figures, the entropy can vary from as little as 12-13 bit to up to 203 bits. Even large 512-bit frames can carry as little as 61 bits of entropy while a smaller 256-bit frames can carry almost the same amount at 60 bits of entropy. In the extreme case, frames that carry 0 entropy, i.e., identical data on all frames from the same ID were also present in the real-world trace. In total, there were 15 such frames. Perhaps surprising, this happens even for two of the larger frames of 512 bits. We emphasize that this is the minimum entropy that was recorded on a trace of $\approx$ 1 million packets but there is no guarantee that it will stay so for the full run-time. Consequently, these values serve merely as a lower

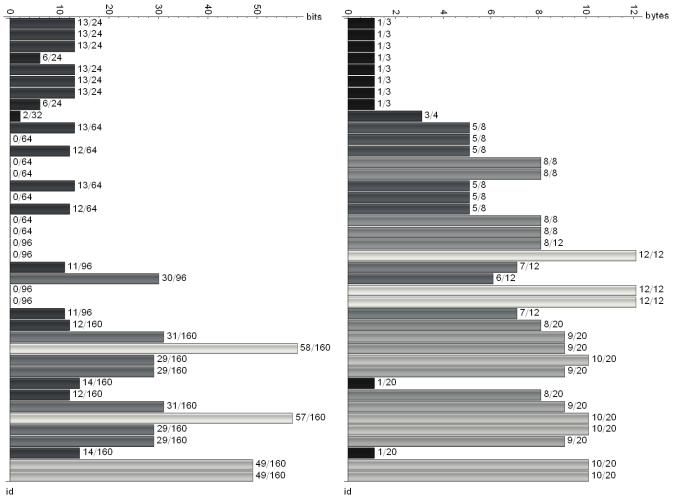

Fig. 7. Barchart of recorded entropies in bits (left) and length of the frame mask in bytes (right) for smaller frames 24–160bits

bound on the number of distinct bits carried by the frame. The byte masks vary accordingly from masks filled with ones or zeros. For frames of zero entropy the length of the byte-mask will be filled with ones since in these frames no change is recorded on the trace.

Finally, in Figure 9 we depict the byte-masks for the 89 IDs in our trace. The size of the frames from Figure 9 varies from 3 bytes to 64 bytes (the maximum allowed by CAN-FD). The black values denote bytes that are unchanged over the entire trace while the light-gray values denote bytes that change at least once. White squares denote unallocated bytes, i.e., for frames smaller than 512 bits.

We performed a similar analysis on the publicly available CAN bus dataset that was used by the authors in [21]. Unfortunately, at the time of our work, the downloaded dataset did not contain a flag to separate between genuine and injected frames in case of fuzzy and impersonation attacks. This made

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2018.2869351, IEEE Transactions on Information Forensics and Security

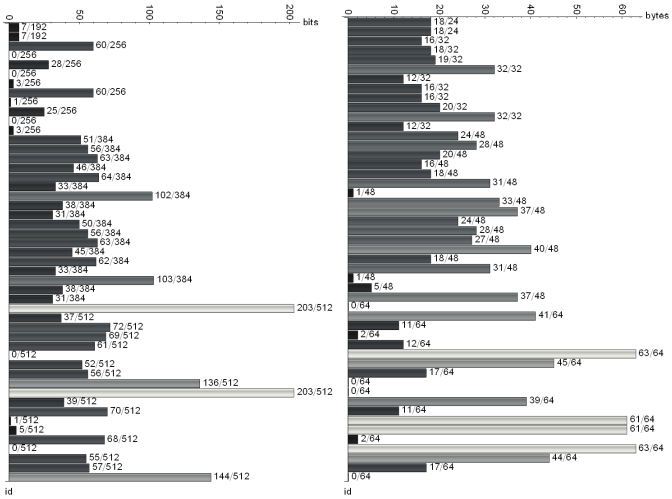

Fig. 8. Barchart of recorded entropies in bits (left) and length of the frame mask in bytes (right) for larger frames 192–512bits

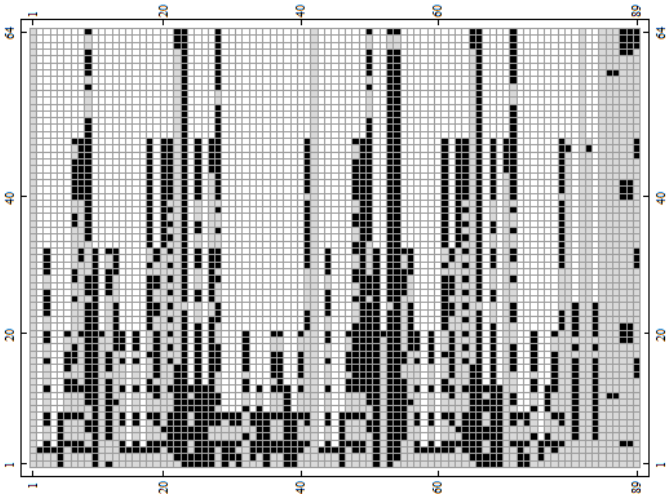

Fig. 9. Graphical depiction for byte-masks for each of the 89 IDs computed from ≈1 million packets recorded on the CAN bus (1 denote by black, 0 denoted by light-gray, no bytes on white)

the dataset unsuitable in our experiments from the next section since it is impossible to make a clear assessment on the sensitivity (true positives) and specificity (true negatives) of the detection mechanism. Still, a brief look at the data made public by the authors in [21] shows that the CAN trace from the Kia Soul vehicle is very similar to the data that we used. In fact, detection might be even simpler since the frames are smaller and carry less data (we measured at most 11 bits of entropy, while our frames top at 203 bits of entropy). In the left side of Figure 10 we depict the timings and content recorded for ID 0x164 in the attack free state. ID 0x164 was the only ID targeted by impersonation attacks. By analyzing the dataset from the attack free state we determined that 6 of the 8 bytes of ID 0x164 are constant - a result obtained by analyzing the first 1,000,000 frames of the attack free state in which 54270 frames had ID 0x164. The right side of Figure 10 depicts the case of fuzzy attacks on ID 0x164, both data modifications and timing deviations up to 3 times the normal 10ms delay of this frame are clearly visible. Data modifications alone

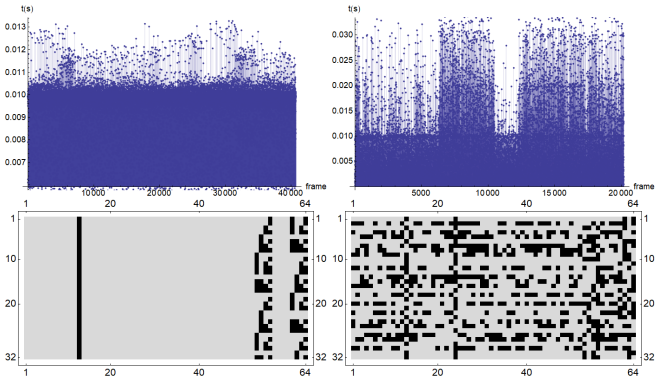

Fig. 10. Depiction of timing and frame content based on data from OTDIS [21] in case of an attack free bus (left) and fuzzy attacks (right) for ID 0x164

will be immediately detected by the masking algorithm while duplicate frames will be reported as replay attacks by our filtering algorithm. The data from the CAN traces in [21] shows similarities with respect to the datasets that we use in terms of periodicity of the messages and the fact that many data fields are fixed and can be easily used to check for anomalies. Further, the specific frame content that we use in the experiments is more complex and has higher entropy which likely makes our dataset a better testbed.

III. Proposed Intrusion Detection Algorithm

We start with a brief outline of the adversary model and outline some Bloom filtering basics. Then we proceed to the description of the proposed intrusion detection algorithm with cascade Bloom filters.

A. Adversary model

Our adversarial model accounts for two actions which we consider to be the most relevant actions of an intruder: replay and modification attacks. In case of a replay attack, a frame that is identical to the genuine frame is sent over the bus. For modification attacks, the data-field of the frame is filled with random data. This behaviour is identical with the fuzzy attacks from related work in [21].

For impersonating a target node, the authors in [21] consider that the genuine node is stopped from transmitting and replaced by an adversarial node. Removing a node from the bus requires either physical intervention or placing the node in the bus-off state, the later possibility being recently demonstrated by the work in [4]. To achieve this, the authors in [4] exploited the CAN error management system which generates active error flags on the bus. Active error flags are visible to all other nodes, but indeed these flags may have natural causes and thus they are insufficient to signal an intrusion. However, if the targeted node will go off-line, this will likely result in losing more frames since an ECU is usually responsible of more than a single ID (or functionality in the car) and the error frames along with the lost IDs should provide a good indication that a node is in bus-off. If only part of the IDs from the targeted node resurface on the bus, then such an abnormal behaviour may be sufficient to signal an intrusion. We also note that the bus-off state of a node may be only temporary as the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2018.2869351, IEEE Transactions on Information Forensics and Security

CAN bus specifications allows recovery after 128 occurrences of 11 recessive bits (which at 1Mbps is ~1.5ms), so this attack could be short lived or the adversary will have to repeatedly send error flags on the bus. Consequently, while an adversary that sends nodes to bus-off by injecting error flags is indeed stronger, the resulting behaviour seems to be more conspicuous and it may be easier to detect for an IDS. Finally, if an adversary places a node off-line and then broadcasts all frames that the node is in charge of, with exactly the same timing and content, then any IDS that analyzes frame timing and content alone will likely fail to detect the intrusion. The authors in [37] already demonstrated that even clock deviations of other nodes from the bus can be faked, which makes timing alone a fragile indicator. The solution to this problem is to rely on cryptography and secret keys that are not available to the adversary, but this is not subject of an intrusion detection mechanism based on traffic analysis. Our IDS provides only some degree of protection in this case, namely, if random modifications are done these are detected by the masking algorithm regardless of the timing at which the frame is sent. Extending our approach to record error flags or check whether certain IDs did not occur on the bus is possible, but it would have only complicated the exposition in this work, a reason for which we do not account this behavior in our analysis.

In [21] DoS attacks are also considered for intrusion detection. In this type of attack an adversary sends the highest priority ID 0x000 and thus causes a DoS on the bus. We omit to specifically address this scenario since detection is trivial as the ID 0x000 does not occur in normal runs. Also, considering only ID 0x000 as cause for DoS as done in [21] may not be sufficient since any other higher priority ID will cause the same problems. If such higher priority IDs do not occur in normal traffic runs or occur more often than usual (a behaviour consistent with replay attacks) the proposed intrusion detection scheme will detect this as a replay. Indirectly, without specifically accounting for DoS attacks, our proposal will detect such intrusions.

B. Bloom filtering basics

Bloom filters [2] are a probabilistic structure for testing membership in a set with 100% recall rate. That is, while false positives may be reported by the filter, there are no false negatives. For addressing false positives, several improvements were proposed: the use of complement filters in [24] or the addition of a secondary filter for treating false positives in [22].

The Bloom filter is an m bit array accompanied by k distinct hash functions. When passing a message through the filter, the message is passed through the k hash functions that activate k distinct positions inside the filter - these positions are set to 1. Testing for membership requires hashing the message through the k hash functions and checking that the corresponding positions are all set to 1 inside the filter.

Given an m-bit filter and n messages that will pass through the filter, the optimal number of hash functions is $k = mn^{-1}ln2$. Given the false positive probability p, the filter size m can be computed as: $m = -nlnp(ln2)^{-2}$. Note that while the filter size m increases with the number of elements n, the number of bits per element m/n is constant given a fixed false positive probability p. In Figure 11 we show the variation in the number of bits per element (left) and number of required hash functions (right) with false positive probability p. Even at a very low false positive probability of 10^{-3} the number of required bits/element is 15 which is lower than the 29 bits required for storing each ID explicitly in case of extended frames. While the number of IDs is generally much lower than the maximum address space and storing them is feasible, the procedure easily extends to the rest of the data-field, a case in which Bloom filtering will be even more effective as a time-memory trade-off. In particular, rather than storing the full data-field, we merely store 1 bit for each byte which determines if the corresponding byte is used in the filtering process. The number of hash computations is less than 10 even if a low false positive probability of 10^{-3} is desired.

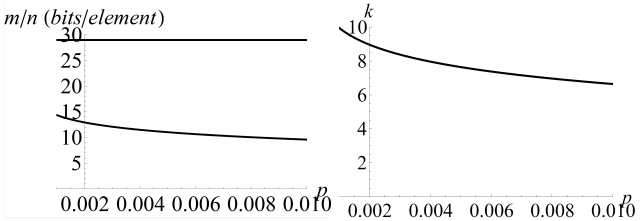

Fig. 11. Variation in bits/element (left) and hash computations (right) with p

C. The cascade filtering mechanism and protocol

We define a *cascade Bloom filter* as a collection of Bloom filters $\phi = \{\varphi_1, \varphi_2, ..., \varphi_\ell\}$ associated to time intervals $\Delta = \{\delta_1, \delta_2, ..., \delta_\ell\}$. A *learning stage* is used to calibrate the filters in ϕ such that they recognize message periodicity based on its ID. We assume that the learning stage is done in a safe environment during production time and there is no adversarial activity on the bus. The learning stage of the cascade filtering algorithm is graphically depicted in Figure 12 and formalized in Algorithm 1 from Figure 14. The detection stage is graphically depicted in Figure 13 and formalized in Algorithm 2 from Figure 14. We explain these in what follows.

In the learning stage, see Figure 12, for each filter $\varphi_i, i \in 1..\ell$ we use a pre-filtering phase immediately followed by a fix-filter phase. The role of the pre-filter is to collect the IDs that arrive during a time interval of length $\delta_i + \epsilon/2$, here ϵ is a small margin to compensate for delays on the bus. The pre-filter φ_{pre} is reset before the learning periods of each filter. By Θ_i^{pre} and Θ_i^{fix} we denote the time intervals for the pre-filtering and fix-filter phase respectively. Consequently, the pre-filter φ_{pre} is updated with all packets arriving in the *pre-filtering* period, i.e.,

$$\Theta_i^{pre} = \left[2\sum_{j=1}^{i-1}\left(\delta_j + \frac{\epsilon}{2}\right), 2\sum_{j=1}^{i-1}\left(\delta_j + \frac{\epsilon}{2}\right) + \delta_i + \frac{\epsilon}{2}\right) \quad (1)$$

Since each filter j has a learning period of $2\delta_j + \epsilon$, the pre-filtering stage of filter i starts at $\sum_{j=1}^{i-1}\left(2\delta_j + \epsilon\right)$ (this is the sum of the learning time for each of the previous filters).

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2018.2869351, IEEE Transactions on Information Forensics and Security

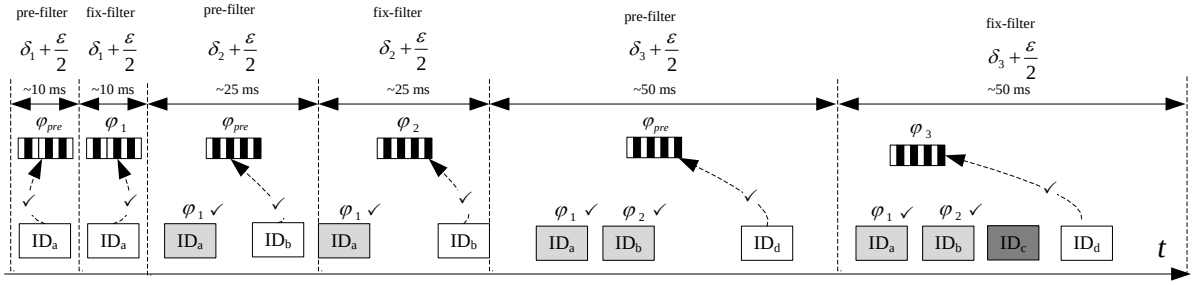

Fig. 12. Cascade Bloom filtering: the learning stage

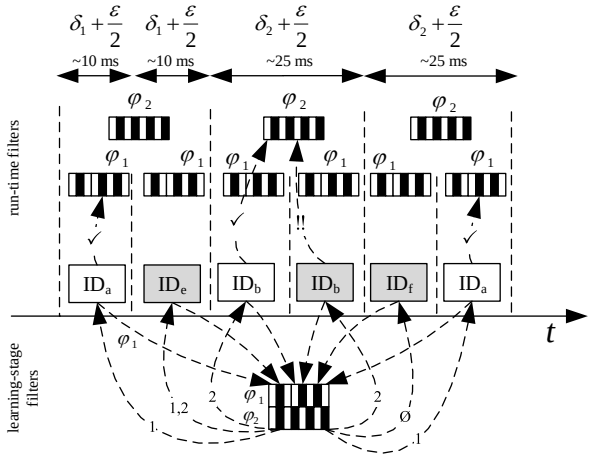

Fig. 13. Cascade Bloom filtering: the run-time stage

Subsequently, each of the filters is set based on messages arriving in the *fix-filter* period, i.e.,

$$\Theta_i^{fix} = \left[2\sum_{j=1}^{i-1}\left(\delta_j + \frac{\epsilon}{2}\right) + \delta_i + \frac{\epsilon}{2}, 2\sum_{j=1}^{i}\left(\delta_j + \frac{\epsilon}{2}\right)\right) \quad (2)$$

This interval starts at the end of Θ_i^{pre} and ends in time $\delta_i + \frac{\epsilon}{2}$. The full learning period is the sum of all learning intervals plus the pre-filtering stage, i.e.,

$$T_{learn} = \sum_{i=1,\ell}\left(\Theta_i^{fix} + \Theta_i^{pre}\right). \quad (3)$$

We now give specific details on the learning and detection algorithms presented in Figure 14. External functions that are called in the algorithms (marked by straight fonts) are performing what their names suggest, e.g., BitOr, BitAnd, BitXor are the corresponding binary operators, Length and Append are the corresponding string operators, GetTime, GetID, GetFilterIndex return time, the ID of the CAN frame and the index of the filter for the current interval. GetMaskedData(*frame*) returns the data-field parsed according to the masking data, i.e., the ones and zeros from the mask determine whether a byte from the data-field is preserved or set to zero, i.e., unused.

Learning algorithm. Algorithm 1 is immediately called whenever a frame is received. In steps 2–6 the current time t is retrieved, the frame identifier *id* is extracted, the data *data* is extracted, masked and concatenated to the identifier *id*, then the index of the current filter *ind* is computed based on the value of t according to the definition of Θ_i^{pre} and Θ_i^{fix}. The index *ind* is set to -1 if we are in the pre-filtering phase. In steps 7–11 φ_{pre} is cleared and the flag clr_{pre} is set denoting that φ_{pre} was cleared. This prepares the fix-filter phase and φ_{pre} needs to be cleared as soon we enter a new pre-filtering phase, otherwise if we are in the fix-filter phase the flag is set to false. In steps 12-16 we check if the retrieved *id* has hit any of the previous filters $j = 1..ind - 1$. By $\varphi(data)$ we denote the result obtained from applying the filter hash functions over the data-field. If this is not the case, i.e., $\neg hit$, we are either in the pre-filtering stage (step 18) and the frame is learned by φ_{pre} or else (step 19) if the packet pases φ_{pre} (step 20) it is also added to φ_{ind} (step 21). In this way, only packets that did not already pass any of the previous filters $\varphi_j, j = 1..i-1$ are allowed in φ_{pre} (this is tested in steps 12–16) and φ_i is updated (step 21) only with packets that passed through φ_{pre} and none of $\varphi_j, j = 1..i-1$ (tested in steps 12–16). This ensures that any packet that goes through φ_i occurred twice during a period of $2\delta_i$ but never occurred twice in the smaller intervals $2\delta_j, j = 1..i-1$.

The learning stage is also graphically illustrated in Figure 12. In this figure the IDs are depicted as they arrive on the bus. First ID_a arrives and this is learned by pre-filter φ_{pre}. Then the ID_a arrives again and since it is recognized by φ_{pre} it is also learned by filter φ_1. When ID_a arrives for the third time, it is recognized by φ_1 and since has been already classified, it will not be learned by the new instance of φ_{pre}. Next, ID_b arrives for the first time and it is learned by φ_{pre}. Then again ID_a arrives and it is recognized by φ_1 and ID_b which now learned by φ_2 since it is recognized by φ_{pre}. In all subsequent occurrences of ID_a and ID_b these are recognized by φ_1 and φ_2 and will be ignored by φ_{pre}. Finnaly, ID_d is learned by φ_3. When ID_c arrives on the bus, it is not recognized by φ_{pre} since it did not occur previously and thus it will not be learned by φ_3.

Intrusion detection. The intrusion detection in Algorithm 2, is actually straight forward. Steps 2–6 are identical to the learning phase algorithm presented previously. If any frame hits more than 1 filter (we verify this in steps 6–14) an

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2018.2869351, IEEE Transactions on Information Forensics and Security

intrusion is signalled (step 13), with properly chosen filter size this event should however have low probability. In steps 15–18 we keep a run-time filter for each interval that is refreshed at each iteration for the corresponding interval (the number of the current iteration is computed in step 16). In step 21 frames that appeared more than once in the run-time stage are detected as intrusions while in step 22 frames that occurred for the first time are learned by the current filter. The intrusion detection algorithm ensures that a frame learned at periodicity δ_i does not occur more than twice in such interval.

The run-time stage, which uses the intrusion detection algorithm, is also graphically illustrated in Figure 13. In this figure the IDs are depicted as they arrive on the bus. First $\mathrm{ID_a}$ arrives and this is verified by φ_1 and updated to the run-time filter φ_1. Then $\mathrm{ID_e}$ but a filter collision occurs at this is checked by two filters φ_1 and φ_2 thus an intrusion is signalled (the filter collision event is for illustration purposes only, by proper choice of filter size this event is unlikely). Subsequently, $\mathrm{ID_b}$ arrives and this again passes as valid message similar to $\mathrm{ID_a}$. Then $\mathrm{ID_b}$ occurs again, it is recognized by φ_2 but the run-time filter φ_2 has already set its bits and thus this is detected as an intrusion (replay attack). Then $\mathrm{ID_f}$ arrives which is not recognized by any of the filters, thus an intrusion is signalled. Finally, $\mathrm{ID_a}$ arrives which is again in order.

IV. Experimental results

In this section we provide practical results for the efficiency of the filters based on recorded CAN bus traces and computational results on real-world power-train ECUs (Electronic Control Units).

A. Rationale behind our experiments

In all scenarios that follow, the run-time analysis was carried for a trace of 100,000 frames. We present the results both after the analysis of the entire trace and in the middle of the trace, i.e., at frame 50,000. Byte-masks were generally computed at 25,000 frames which proves to be a good predictor for the next 100,000 frames. The learning stage, i.e., the length of the trace required for training the filters, depends on the size of the filters and is generally a few thousand packets in length. The number of adversarial injections is available from the table containing the results by summing the true positives with the false positives. In general, for 100,000 genuine frames there are 1,000 adversarial frames injected at random locations. If the result is for the middle of the trace, there are ~500 adversarial frames since the 1,000 injections are uniformly distributed in the trace. We choose to keep the number of injections at 1,000 frames for 100,000 genuine frames since by increasing the number of injections the attack becomes conspicuous and we assume that in a real-world scenario an adversary would prefer to stay stealthy at least to some extent. The experiments validate several scenarios and hypothesis that we outline next.

Simulating adversarial attacks. The attacks were simulated by inserting adversarial frames in the genuine trace. The insertion point was either immediately after the genuine frame or at some randomized time but no later than the next occurrence of a genuine frame with the same ID. Immediate injection should be easier to detect unless the previous genuine frame was a false-positive (in this case the adversarial frame is mismatch for the genuine frame). Delayed injection should be harder to detect since adversarial frames may be mismatch for genuine frames. We consider both replay and modification attacks. In a replay attack the adversary injects a frame that is identical to the genuine frame. Replay attacks should be in general harder to detect than modification attacks since they carry data that looks normal, it is only the timing that may not match. For the modification attack we considered that the adversary alters the

Algorithm 1 Learning algorithm (learning stage)

```
1: procedure LEARN FRAME
2:     t ← GetTime()
3:     id ← GetID(frame)
4:     data ← GetMaskedData(frame)
5:     data ← Concat(id, data)
6:     ind ← GetFilterIndex(t)
7:     if (ind = −1 ∧ ¬clr_pre) then
8:         φ_pre ← BitXor(φ_pre, φ_pre); clr_pre = true
9:     end if
10:    if (ind ≠ −1 ∧ clr_pre) then clr_pre = false
11:    end if
12:    hit ← false
13:    for j = 1 → ind–1 do
14:        if φ(data) = BitAnd(φ(data), φ_j) then hit ← true
15:        end if
16:    end for
17:    if ¬hit then
18:        if ind = −1 then φ_pre ← BitOr(φ_pre, φ(data))
19:        else
20:            if φ_pre = BitAnd(φ_pre, φ(data)) then
21:                φ_ind ← BitOr(φ_ind, φ(data))
22:            end if
23:        end if
24:    end if
25: end procedure
```

Algorithm 2 Filtering algorithm (run-time stage)

```
1: procedure FILTER FRAME
2:     t ← GetTime()
3:     id ← GetID(frame)
4:     data ← GetMaskedData(frame)
5:     data ← Concat(id, data)
6:     flpassed ← {}
7:     for j = 1 → ℓ do
8:         if φ(data) = BitAnd(φ(data), φ_j) then
9:             flpassed ← Append(flpassed, j)
10:        end if
11:    end for
12:    if Length(flpassed) ≠ 1 then
13:        return Intrusion
14:    end if
15:    ind = flpassed[1]
16:    iteration = ⌊(t − t_start)/δ_ind⌋
17:    if iteration > count[ind] then
18:        φ^run_ind = φ(data)
19:        count[ind] = iteration
20:    else
21:        if φ(data) = BitAnd(φ^run_ind, φ(data)) then return Intrusion
22:        else φ(data) = BitOr(φ^run_ind, φ(data))
23:        end if
24:    end if
25:    return ¬Intrusion
26: end procedure
```

Fig. 14. Learning and intrusion detection by Bloom filtering on CAN frames

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2018.2869351, IEEE Transactions on Information Forensics and Security

frame with some random noise. Modification attacks should be easier to detect when masks are used.

Byte masks and filter training. Byte-masks are more effective when dealing with modification attacks, otherwise, they slightly increase the false-positives rate. On the contrary, disabling the byte-masks lowers the false-positives rate (frames of high entropy hold higher risks to be reported as false-positives due to significant modification of their bytes). However, disabling the byte-masks will increase the false-negatives rate. The training duration has clear impact on the filter performance. The length of the training refers to both filter calibration and the extraction of byte-masks. The longer the training duration the better the detection. However, the longer the training trace for the masks, the fewer of the bytes will be used as masks are filled with zeros. Thus choosing and optimal training length is decisive. By experiments, we determined that in general k frames are a good predictor for the next $4k$ frames, but this of course varies according to the data from the trace. We choose to work with binary byte masks which either take or drop a byte since no specific information on the frame content was available to us. If additional information exists on how signals change inside the data-field (e.g., for speed, torque or other predictable values), then the byte masks can be adapted by first mapping the data through a specific function for each signal with a known behaviour.

Filter calibration. The success rate can be tuned by modifying the parameters of the Bloom filters, i.e., the size of the filter and the number of hashes. Generally, we tested the approach for 10 hash computations and 1024 bit filters which seem to be a good choice for increased accuracy. Good results were obtained when we lowered these to 9 hash functions and 512 bits, then for 7 hash functions and 256 bits some degradation was obvious. The size of the filters clearly depends on the length of the training trace. Selecting specific time-intervals is critical for the detection rate. For the J1939 simulation we had direct information on the timing of each frame while for the CAN-FD trace (since we had no additional information) we took a distinct approach by testing several timings for the filters. The timings for CAN-FD frames were not hard to guess as they were quite close to those from J1939 simulation.

B. Results on a power-train bus for commercial-vehicles

We first employ an existing J1939 CANoe simulation to obtain experimental data. The setup consists of 6 nodes responsible for basic power-train functionalities which communicate according to J1939 - a CAN-based higher layer protocol for commercial vehicles. The CAN messages used within the simulation correspond to 38 cyclic IDs (with cycles between 10 ms and 1s) and several on-event frames. This is in-line with existing real-world data, for example in [5] 39 cyclic IDs are reported for a Toyota Camry and 55 for a Dodge RAM.

We used an array of seven filters, i.e., $\phi = \{\varphi_1, \varphi_2, ..., \varphi_7\}$, that correspond to time intervals $\Delta = \{0.01s, 0.020s, 0.050s, 0.100s, 0.250s, 0.500s, 1s\}$ according to the delays at which cyclic frames are sent in the simulation. The delays are more stable than in the case of the real-world trace that we discuss in the next section, but still, drifts from the expected arrival time are common as shown in Figure 15.

First, in Figure 15 (a) we illustrate two IDs with arrival time 1s with more delays (left) and less delays (right), the differences in the arrival times are generally less than 0.1%. Then in Figure 15 (b) we show the timings of two IDs with arrival time of 100ms, the plot on the left side shows a more stable arrival time while the one in the right side exhibits more variations. Higher variations are due to an ID with lower priority on the bus. Still, the delays are small. Finally, in Figure 15 (c) we show the timings of two IDs with a periodicity of 50ms and 10ms respectively. On the left side the arrival time is very stable but on the right the arrival time exhibits delays of up to 40%. Such delays are the main factor behind a false-positive reports but can be alleviated by a better allocation of IDs and traffic on the CAN bus (this is however out of reach for our work).

On the recorded trace, two kinds of attacks were simulated by injecting frames in the trace: replay attacks which consist in injecting and identical frame and modification attacks which consist in injecting a frame with a random data-field. The resulting log file is then parsed by the cascade Bloom filtering procedure which is implemented in a high-level language. Processing the trace file in a high-level language should not cause any concerns about the real-world implementation since the CANoe trace is exactly what each node will record from the CAN bus. To prove feasibility from a computational point of view, we provide computational results for hash functions on automotive-grade controllers at the end of this section.

We present the results in Table I. The learning period was of 1759 frames which is around 3-4 seconds, the length is fixed by the structure of the filters, i.e., $\Delta = \{0.01s, 0.020s, 0.050s, 0.100s, 0.250s, 0.500s, 1s\}$. To cover sufficient modifications at byte-level, masks were extracted over a longer period of 25,000 frames which roughly corresponds to 1 minute of run-time. When the attack immediately follows after the genuine frame it is immediately detected with a false-negative rate of 0%, i.e., lines (1) and (2) from Table I. A false-positive rate of 5-6% is present due to the existing delays from the bus but this happens only for IDs with lower priority. Once the replay attack is done at a randomized delay, the false negative rate gets to 34-35%, i.e., lines (3) and (4) from Table I. While on a first view this may seem as poor performance in detecting the attack, this is not necessarily so because what happens is that adversarial frames are now replacing the genuine frames as the number of false-positives increases. That is, while on lines (1) and (2) from Table I we had 2908 and 6063 false-positives for 50,000 and 100,000 frames respectively, we now have 3082 and 6352 false-positives. This means that the intrusion can still be signaled since the filter detects more frames than expected, but it is not possible to discern between genuine and replay frames.

The case of modification attacks leads to 0% of false-negatives since the data-field of the frame does not pass from the Bloom filter (Bloom filters have a 100% recall rate). This happens regardless of the random delays that are introduced in the transmission of the frames and thus the results from line

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2018.2869351, IEEE Transactions on Information Forensics and Security

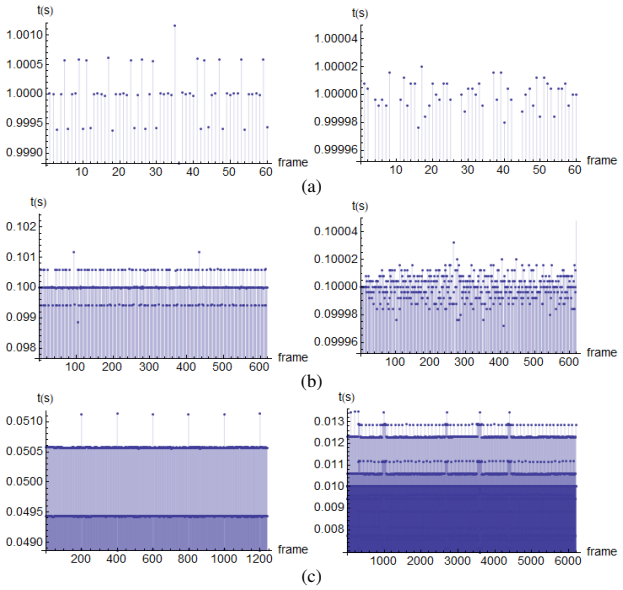

Fig. 15. Differences in frame arrival times on CANoe J1939 simulation: two ids with arrival time 1s with more delays (left) and less delays (right) (a), two ids with arrival time of 100ms with less delays (left) and more delays (right) (b), two ids with small arrival time 50ms (right) and 10ms (left) (c)

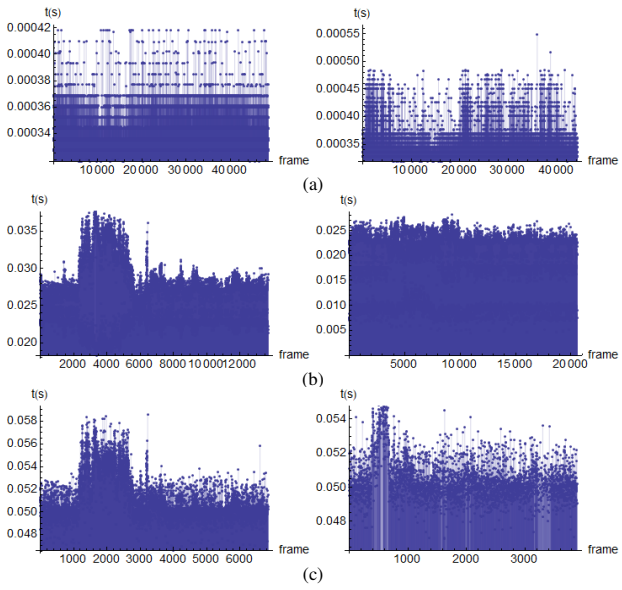

Fig. 16. Differences in frame arrival times on recorded in-vehicle traffic: two short and irregular arrival time of 300 μs (left) and 400 μs (right) (a), normal arrival time of 25ms with some delays (left) and no delays (right), normal arrival time of 50ms with some delays (c)

TABLE I
FILTERING SUCCESS RATES FOR THE J1939 SIMULATION ($\Delta = \{0.01s, 0.020s, 0.050s, 0.100s, 0.250s, 0.500s, 1s\}$)

	Attack params.		Training and testing				Bloom filter params.		Results					
No.	Attack type	Delayed attack	Byte mask	Learning (frames)	Masking (frames)	Testing (frames)	Hashes (k)	Size (m) bits	FPR	FNR	TN (frames)	TP (frames)	FP (frames)	FN (frames)
(1)	r	no	yes	1759	25000	50000	10	1024	0.05	0.	46622	470	2908	0
(2)	r	no	yes	1759	25000	100000	10	1024	0.06	0.	93937	1000	6063	0
(3)	r	yes	yes	1759	25000	50000	10	1024	0.06	0.34	46409	332	3082	177
(4)	r	yes	yes	1759	25000	100000	10	1024	0.06	0.35	93583	646	6417	354
(5)	m	no	yes	1759	25000	50000	10	1024	0.05	0.	46622	470	2908	0
(6)	m	no	yes	1759	25000	100000	10	1024	0.06	0.	93937	1000	6063	0
(7)	m	yes	yes	1759	25000	50000	10	1024	0.05	0.	46586	509	2905	0
(8)	m	yes	yes	1759	25000	100000	10	1024	0.06	0.	93937	1000	6063	0
(9)	r	no	yes	1759	25000	100000	9	512	0.06	0.	93937	1000	6063	0
(10)	m	yes	yes	1759	25000	100000	7	256	0.06	0.002	93937	998	6063	2

(5) to (8) are identical. The J1939 simulation scenario proved not to be very demanding and we successfully reduced the filters to 512 bits and 9 hash functions with no performance degradation as can be seen in line (9) of Table I. By going even lower to 256 bits and 7 hash functions 2 false negative messages appeared in case of the delayed modification attack, this is shown in line (10) of Table I.

C. Result on a real-world CAN-FD trace

The experiments conducted on a CAN-FD trace from a real-world high-end vehicle are more demanding as we now discuss.

In Figure 16 we depict differences in arrival times for six distinct IDs. First in Figure 16 (a) we show data for two IDs that are harder to manage by the filtering algorithm. In this case the arrival time is very short at around 300 μs and variations of around 50% are common. Fortunately, these two IDs are the only (out of the 89 IDs from the trace) that cause such problems. Then in Figure 16 (b) and (c) we show data for IDs with arrival time of 25ms and 50ms respectively. For the IDs on the left side of Figure 16 (b) and (c) delays have a similar impact on the arrival time, likely due to a very close value of the identifiers which leads to identical priority. The right side of Figure 16 (b) and (c) shows IDs with the same periodicity but which are affected by delays to a lesser extent (due to a higher priority of the ID). These delays are the main factor behind the false positive reports of the filter, they can be alleviated by better allocation of the traffic on the bus but this is out of reach for our work.

In Table II we summarize the experimental results. The length of the trace for filter calibration is always 4208 frames, this is due to the fixed periodicities that were taken into account by the filters, i.e., $\Delta = \{30\mu s, 25ms, 50ms, 1s\}$. When choosing the length of the trace for computing the message masks it proved that using the same length as for the learning stage results in satisfactory results only for the next few thousand frames, then the performance degrades for the false-positives rates. This is due to the high entropy of the genuine frames which can be mismatch as adversarial actions due to significant modifications. We then choose a longer trace of 25,000 packets for the mask which proved a good predictor for the next 100,000 frames. This suggests that regular updates

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2018.2869351, IEEE Transactions on Information Forensics and Security

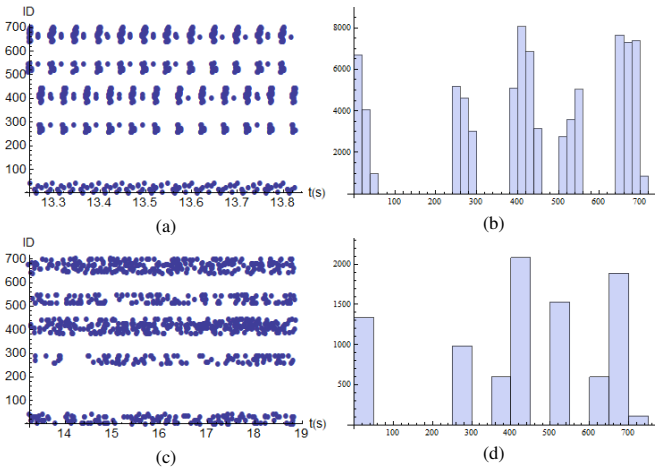

Fig. 17. Graphical depiction of recorded time (left) and histogram distribution (right) for true negatives (up) and true positives (down)

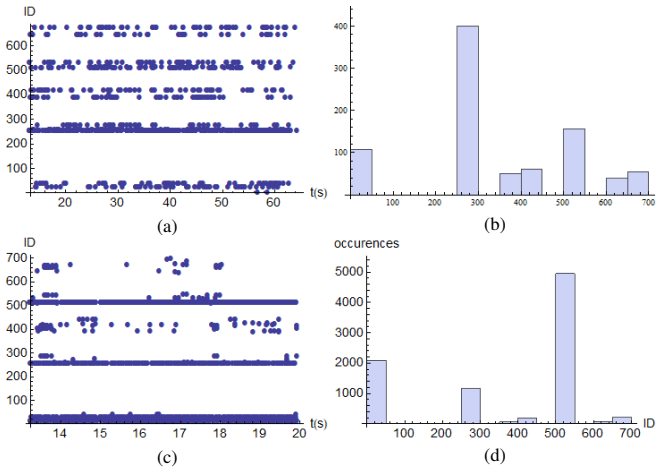

Fig. 18. Graphical depiction of recorded time (left) and histogram distribution (right) for false negatives (up) and false positives (down)

of the packet mask may be essential for lowering the false-positives rate, we give later a brief discussion on updating the byte masks. Since the usefulness of byte-masks was already proved by the previous experiments, we restrict the analysis that follows only to the case when byte-masks are used.

In rows 1–4 of Table II we present results for modification attacks. When there are no delays in the adversarial frames, i.e., forged frames are inserted next after the genuine frames, the false-negative rate is as low as 1–2%. While the false positive rate tops at 11-17%, only two IDs are responsible for most of the false-positive rate. In case of delayed injection of modified frames, the false-negative increases to 8–9% and the false-positive rate remains the same. For replay attacks, presented in rows 5–8 of Table II, with no delays there is only a slight increase of the false-negative rate to 2–3%. This is expected as in the case of replay attacks byte-masks will not be so effective in detecting the attack. With delayed injection the false-negative rate surges to 43–47%. Again, this should not necessarily be interpreted in a negative sense, the problem is that now genuine frames cannot be separated from adversarial frames and the number of false positives increases from 17,006 to 17,383 (at 100,000 genuine frames). This happens because adversarial frames that arrive before the genuine frames will become a false-negative and the genuine frame will be the false positive. Thus, the filtering algorithm will detect anomalies on the traffic but it cannot discern between genuine frames and adversarial frames. This points out that besides the intrusion-detection algorithms, means of identifying genuine frames must be in place, e.g., cryptographic authentication.

We now discuss the results in Table III for the case when the filter size and number of computations are lowered. We discuss performance degradation only for the best case of modification attacks with immediate injection, for the rest of the cases the results scale up with the corresponding detection rates. When lowering the number of hash computations from 10 to 9 and the filter from 1024 to 512 bits, there is a visible increase in the number of false positives to 16–17% and false-negatives to 4–5%. While the performance degradation is obvious, these values may still be acceptable when resources are limited. Reducing to 7 hash functions and 256 bits, does however increase the false-positive rate to 27–31% and the false-negative rate to 17–19% which suggests that this filter size is inappropriate for the higher data-rate of CAN-FD.

For a better understanding on how true-negatives and true-positives values build up, in Figure 17 we give a graphical depiction of frame arrival time for each of the IDs. For this plot, the ID is converted from decimal representation and ploted on the vertical axis. To avoid overloading the plot, only the first 1000 true-positives and true-negative values are added. The left side of the figure depicts the arrival time and the right side the histogram distribution. It is easy to see that for true-negatives (up) the arrival time follows a clear cyclic pattern. For the true-positives, i.e., adversarial frames detected by the intrusion detection mechanism, the arrival time is random and uniformly distributed over the IDs. This is consistent with the randomized injection attack that was tested in our scenario.

The false-negatives and false-positives are shown in Figure 18. Again, the left side of the figure depicts the arrival time and the right side the histogram distribution. In both cases the values are neither cyclic nor uniformly distributed. Most of the misclassified frames tend to group around IDs that exhibit significant drifts from the expected arrival time are both more easy to pass as false-positives (down) and false-negatives (up). Adversarial frames and genuine frames are more likely to be mismatch for IDs with lower priority that are sent at a higher rate.

Re-computing byte-masks. As proved by our experiments, byte-masks are very effective in increasing the detection of modified frames but they do also increase the number of false-positives when they are used for longer periods. For this purpose, in a real-world deployment, they may be periodically updated. Updating a byte-mask for a particular ID will require keeping a copy of the data-field of the most recent packet. Once a new packet arrives, bytes that are distinct are updated and the mask bit is set to 0. A more in-depth analysis can help in deciding the optimal choice of byte-masks and the intervals at which they should be updated but this is out of scope for the current work.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2018.2869351, IEEE Transactions on Information Forensics and Security

TABLE II
FILTERING SUCCESS RATES FOR FRAME MODIFICATION AND REPLAY ATTACKS ON A CAN-FD TRACE ($\Delta = \{30\mu s, 25ms, 50ms, 1s\}$)

	Attack params.		Training and testing				Bloom filter params.		Results					
No.	Attack type	Delayed attack	Byte mask	Learning (frames)	Masking (frames)	Testing (frames)	Hashes (k)	Size (m) bits	FPR	FNR	TN (frames)	TP (frames)	FP (frames)	FN (frames)
(1)	m	no	yes	4208	25000	50000	10	1024	0.11	0.01	43574	479	5938	9
(2)	m	no	yes	4208	25000	100000	10	1024	0.17	0.02	82996	975	17004	25
(3)	m	yes	yes	4208	25000	50000	10	1024	0.12	0.09	43548	431	5974	47
(4)	m	yes	yes	4208	25000	100000	10	1024	0.17	0.08	82942	911	17058	89
(5)	r	no	yes	4208	25000	50000	10	1024	0.11	0.02	43573	475	5939	13
(6)	r	no	yes	4208	25000	100000	10	1024	0.17	0.03	82994	970	17006	30
(7)	r	yes	yes	4208	25000	50000	10	1024	0.12	0.47	43375	249	6147	229
(8)	r	yes	yes	4208	25000	100000	10	1024	0.17	0.43	82617	570	17383	430

TABLE III
FILTERING SUCCESS RATES WITH REDUCED NUMBER OF HASHES AND FILTER SIZE ON A CAN-FD TRACE ($\Delta = \{30\mu s, 25ms, 50ms, 1s\}$)

	Attack params.		Training and testing				Bloom filter params.		Results					
No.	Attack type	Delayed attack	Byte mask	Learning (frames)	Masking (frames)	Testing (frames)	Hashes (k)	Size (m) bits	FPR	FNR	TN (frames)	TP (frames)	FP (frames)	FN (frames)
(1)	m	no	yes	4208	25000	50000	9	512	0.16	0.04	41418	464	8094	24
(2)	m	no	yes	4208	25000	100000	9	512	0.20	0.05	79048	950	20952	50
(3)	m	no	yes	4208	25000	50000	7	256	0.27	0.17	35881	401	13631	87
(4)	m	no	yes	4208	25000	100000	7	256	0.31	0.19	68773	807	31227	193

D. Computational results

The analysis from the previous section clarifies the performance of the proposed intrusion-detection mechanism. For deployment in a real-world vehicle, computational constraints are the main limitation. Memory requirements are not high at one bit for each byte of an ID, which given a frame length of at most 512 bits for CAN-FD results in at most 64 bits for each ID plus the ID itself. As already stated, previous work in [5] accounted for at most 55 IDs on the network, the high-end vehicle trace that we analyzed had 89 IDs and CAN-FD frames. But even for this extreme case, only a few of the IDs carried the 64 bytes allowed by the standard and the 89 IDs sum up to 2920 bytes. Since each byte requires 1 bit, it leads to around 365 bytes for the masks which should cause no memory issues. In what follows we clarify that from a computational point of view it is feasible to implement Bloom filters on automotive-grade microcontrollers.

To achieve this, we evaluated the computational performance of three representative 32-bit platforms from the automotive domain. On each of them we provide measurements for the time needed for the message to pass the Bloom filtering algorithms. For two of the selected platforms, namely the NXP MPC5606B and AURIX TC297, we used existing development boards while for the Renesas RH850/E1x-FCC1 the tests were done using a simulator. Figure 19 depicts our experimental setup which includes the J1939 CANoe simulation, the Freescale/NXP MPC5606B and the Infineon TC297 development boards.

Table IV summarizes the results obtained for the various tested scenarios. Separate variants of the filter algorithm based on the MD5 and the MurMur hash functions were used. While MD5 offers only weak cryptographic security, we emphasize that its weaknesses (i.e., collision attacks) are not relevant for Bloom filtering. MurMur [1] is not a cryptographic hash and it is insecure from a cryptographic perspective, but it is commonly used for non-cryptographic applications in various platforms. To avoid any suspicions we tested several traces with both MD5 and MurMur and there was no difference of the false-positive and false-negative reports.

Similar to the analysis in the previous section, three variants of Bloom filters were tested using a filter size of 256, 512 and 1024 bits along with the application of the corresponding hash for 7, 8 and 9 times respectively. For each of these variants we applied messages with payloads of 8, 16, 32 and 64 bytes as inputs and measured the time needed to evaluate the filter on the message. We depict the time required both for checking that a message has already passed through the filter column (i.e., the Check column in Table IV) and the time needed to add the message to the filter (i.e. the Set column in Table IV). The Check operation corresponds to computing the filter function φ over id-message pair and the BitAnd operations from Algorithms 1 and 2, e.g., $\varphi_{pre} = \mathsf{BitAnd}(\varphi_{pre}, \varphi(data))$. Similarly, the Set operation corresponds to computing the filter function φ over the id-message pair and the BitOr operations from Algorithms 1 and 2, e.g., $\varphi_{ind} \leftarrow \mathsf{BitOr}(\varphi_{ind}, \varphi(data))$. Note that the Check algorithm is probabilistic and the running time for a message that is recognized by the filter is the same as for Set while for a message that is not recognized the time may vary from a single hash computation (more likely) to all the required hash computations (a less likely situation). The time for the Set function is deterministic since always the same number of hashes is computed. Of course, the runtime of the Set function depends on the size of the input which differs according to the size of the frame. Two additional bytes, representing the message ID, are always added to the message payload.

When using MD5, the execution time for the filter application is less than the transmission of an 8 byte CAN frame at 500kbps with the exception of MPC5606B which exceeds this for almost all tested variants. In the case of MurMur most of the results on the three platforms fit in the aforementioned interval. However, there are still several cases in which the MPC5606B platform exceeds the frame transmission time

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2018.2869351, IEEE Transactions on Information Forensics and Security

TABLE IV
FILTER RUN-TIME ON THREE AUTOMOTIVE PLATFORMS: FREESCALE/NXP MPC5606B, INFINEON TC297, RENESSAS RH850/E1X-FCC1

Platform	Filter size	Hashes	MD5								MurMur							
			64 bytes		32 bytes		16 bytes		8 bytes		64 bytes		32 bytes		16 bytes		8 bytes	
			Check (μs)	Set (μs)	Check (μs)	Set (μs)	Check (μs)	Set (μs)	Check (μs)	Set (μs)	Check (μs)	Set (μs)	Check (μs)	Set (μs)	Check (μs)	Set (μs)	Check (μs)	Set (μs)
MPC5606B	256	7	357.2	2466	1220	1426	1422	1426	208.2	1426	47.17	294.4	59.72	195.2	104.5	145.3	37.98	120.3
	512	9	3168	3168	1220	1832	1827	1832	208.2	1832	378.8	376.6	59.72	249.2	25.49	185.0	37.99	152.9
	1024	10	357.2	3520	410.4	2034	208.2	2034	814.8	2034	47.17	418.0	59.72	276.2	25.49	204.9	37.99	169.2
TC297	256	7	10.53	73.74	35.33	41.20	41.22	41.20	5.90	41.20	3.100	21.74	3.630	12.90	5.760	8.040	1.670	5.740
	512	9	94.80	94.80	35.31	52.95	52.95	52.95	5.88	52.95	27.97	27.93	3.618	16.18	1.158	10.30	1.660	7.360
	1024	10	10.53	105.3	11.78	58.83	5.88	58.83	23.55	58.83	3.116	31.03	3.616	17.97	1.157	11.43	1.660	8.160
RH850/E1x-FCC1	256	7	13.00	86.55	45.45	52.96	52.93	52.96	7.731	49.85	1.993	12.99	2.475	8.259	4.228	5.906	1.446	4.753
	512	9	111.0	111.1	45.44	67.86	67.79	67.86	7.731	63.87	16.58	16.65	2.475	10.54	0.981	7.534	1.443	6.050
	1024	10	12.99	123.3	15.65	75.28	8.200	75.28	30.54	75.28	1.990	18.43	2.468	11.67	0.978	8.290	1.456	6.615

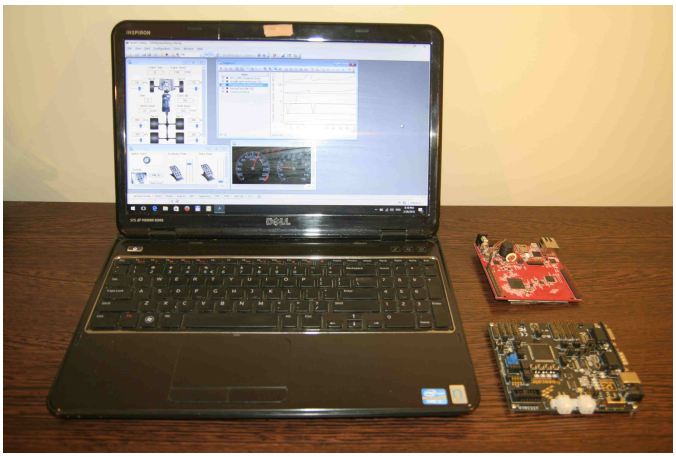

Fig. 19. Experimental setup (J1939 CANoe simulation along the Freescale/NXP MPC5606B and Infineon TC297 developmentd boards)

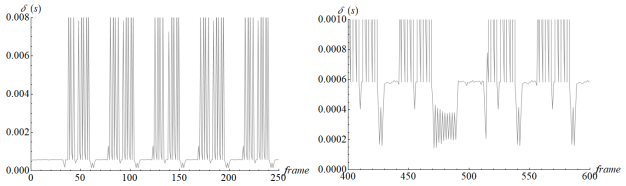

Fig. 20. Regular delays between frames (left) and a burst period (right) from the CANoe J1939 simulation

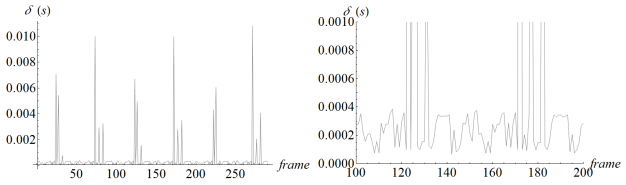

Fig. 21. Regular delays between frames (left) and a burst period (right) from the real-world CAN bus trace

by up to 200μs which would be almost enough to send a second 8 byte CAN frame. This difference in performance is expected since the MPC5606B was clocked at 64MHz in our tests, as it does not support higher frequencies, while the TC297 and RH850/E1x-FCC1 were clocked at 300MHz and 320MHz respectively. We now discuss the relation between computational time and inter-frame delays in each of the two scenarios that were previously evaluated.

By running this CANoe J1939 simulation we obtained an average of less than 500 messages/second, which leads to an average of 2ms of processing time for each message. This is enough time for a high-end controller, e.g., TC297 or RH850/E1x-FCC1. We note however that burst periods occur on the bus as well. Figure 20 depicts delays between packets as recorded in our simulation. On the left side regular delays can be seen to peak at 8ms. Burst periods depicted on the right side have delays of less than 200μs but occur rarely, i.e., they account for less than 5% of the traffic, while the minimum recorded delay is at 144μs. Consequently, for practical reasons, a filter that requires less than 100μs of processing time per message is recommended. Our experimental results show that this is achievable with the TC297 and RH850/E1x-FCC1 controllers but not with the MPC5606B. Still, our view is rather pessimistic since we address the entire traffic on the bus while typically an ECU only receives part of the traffic (due to existing filtering mechanism on CAN). Consequently, for ECUs that cannot cope with the computational demands, filtering only part of the traffic may be an alternative.

Figure 21 shows the delays on the CAN-FD trace. Not surprising, the delays are in general by one order of magnitude smaller. Variations in the inter-frame delays are higher as can be seen in the plot for 300 consecutive frames on the left side of the figure. Also, the variation of the delays during the burst period is less uniform as can be seen on the detailed plot from the right side. The mean and median delays are between 200-500 μs with the minimum recorded delay at 49μs. This generally recommends the use of the non-cryptographic hash MurMur since the cost of MD5, while still feasible, is close to the limit. Again, this evaluation is for the entire trace and it is not expected that an ECU will be responsible for working with the entire traffic from the bus.

V. CONCLUSION

Bloom filters are known to provide a memory efficient mechanism for membership testing. In this work we showed their effectiveness in separating packets on the CAN bus based on their periodicity and content of the data field. Indeed, the timing for the injection of adversarial frames is critical for the false-negative rate of the filtering mechanism. In this respect byte-masks, which allow testing for the content of the frame, are more effective in detecting modified frames. In the worst case, when an adversary can replay frames in the optimal time-frame, duplicate frames will be detected and even if there

is no mechanism to distinguish between genuine frames and adversarial frames an intrusion can be signalled.

We hope that our work opens road for a new applicative area of Bloom filters: automotive grade networks, in particular the CAN-bus but newer technologies such as FlexRay or BroadR-Reach offer identical setups. While a full scale implementation and further improvements are subject of future work for us, by this research we made the first steps for the adoption of such mechanisms on automotive buses which are crucially demanding security.

Acknowledgement. This work was supported by a grant of the Romanian National Authority for Scientific Research and Innovation, CNCS-UEFISCDI, project number PN-II-RU-TE-2014-4-1501 (2015-2017). We thank Eng. Tudor Andreica and Eng. Bogdan Nuna from HELLA Timisoara for providing the real-world CAN-FD traces that were used in our experimental analysis. We are grateful to the anonymous reviewers for their comments which have helped us to improve our work.

REFERENCES

[1] A. Appleby. Murmurhash. *URL https://sites. google. com/site/murmurhash*, 2008.

[2] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[3] 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.

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

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

[6] W. Choi, H. J. Jo, S. Woo, J. Y. Chun, J. Park, and D. H. Lee. Identifying ecus using inimitable characteristics of signals in controller area networks. *IEEE Trans. Vehicular Technology*, 67(6):4757–4770, 2018.

[7] 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.

[8] S. Dario, M. Mirco, and C. Michele. Detecting attacks to internal vehicle networks through hamming distance. In *IEEE 2017 AEIT International Annual Conference-Infrastructures for Energy and ICT (AEIT 2017)*, 2017.

[9] C. E. Everett and D. McCoy. Octane (open car testbed and network experiments): Bringing cyber-physical security research to researchers and students. In *CSET, Presented as part of the 6th Workshop on Cyber Security Experimentation and Test. USENIX*, 2013.

[10] H. Giannopoulos, A. M. Wyglinski, and J. Chapman. Securing vehicular controller area networks: An approach to active bus-level countermeasures. *IEEE Vehicular Technology Magazine*, 12(4):60–68, 2017.

[11] B. Groza, P.-S. Murvay, A. Van Herrewege, and I. Verbauwhede. LiBrA-CAN: a lightweight broadcast authentication protocol for controller area networks. In *11th International Conference on Cryptology and Network Security, CANS 2012, Springer-Verlag, LNCS*, 2012.

[12] B. Groza and S. Murvay. Efficient protocols for secure broadcast in controller area networks. *IEEE Transactions on Industrial Informatics*, 9(4):2034–2042, 2013.

[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] T. Hoppe, S. Kiltz, and J. Dittmann. Applying intrusion detection to automotive it-early insights and remaining challenges. *Journal of Information Assurance and Security (JIAS)*, 4(6):226–235, 2009.

[15] T. Hoppe, S. Kiltz, and J. Dittmann. Security threats to automotive can networks—practical examples and selected short-term countermeasures. *Reliability Engineering & System Safety*, 96(1):11–25, 2011.

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

[17] M.-J. Kang and J.-W. Kang. Intrusion detection system using deep neural network for in-vehicle network security. *PloS one*, 11(6):e0155781, 2016.

[18] M.-J. Kang and J.-W. Kang. A novel intrusion detection method using deep neural network for in-vehicle network security. In *Vehicular Technology Conference (VTC Spring), 2016 IEEE 83rd*, pages 1–5. IEEE, 2016.

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

[20] 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.

[21] H. Lee, S. H. Jeong, and H. K. Kim. Otids: A novel intrusion detection system for in-vehicle network by using remote frame. In *Procedings of PST (Privacy, Security and Trust)*, 2017.

[22] H. Lee and A. Nakao. Improving Bloom filter forwarding architectures. *IEEE Communications Letters*, 10(18):1715–1718, 2014.

[23] H. Li, L. Zhao, M. Juliato, S. Ahmed, M. R. Sastry, and L. L. Yang. Poster: Intrusion detection system for in-vehicle networks using sensor correlation and integration. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 2531–2533. ACM, 2017.

[24] H. Lim, J. Lee, and C. Yim. Complement Bloom filter for identifying true positiveness of a Bloom filter. *IEEE Communications Letters*, 19(11):1905–1908, 2015.

[25] 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.

[26] 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.

[27] M. Marchetti, D. Stabili, A. Guido, and M. Colajanni. Evaluation of anomaly detection for in-vehicle networks through information-theoretic algorithms. In *Research and Technologies for Society and Industry Leveraging a better tomorrow (RTSI)*, pages 1–6. IEEE, 2016.

[28] 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.

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

[30] 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.

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

[32] P.-S. Murvay and B. Groza. Source identification using signal characteristics in controller area networks. *IEEE Signal Processing Letters*, 21(4):395–399, 2014.

[33] M. Müter and N. Asaj. Entropy-based anomaly detection for in-vehicle networks. In *Intelligent Vehicles Symposium (IV), 2011 IEEE*, pages 1110–1115. IEEE, 2011.

[34] M. Müter, A. Groll, and F. C. Freiling. A structured approach to anomaly detection for in-vehicle networks. In *Information Assurance and Security (IAS), 2010 Sixth International Conference on*, pages 92–98. IEEE, 2010.

[35] S. N. Narayanan, S. Mittal, and A. Joshi. Obd_securealert: An anomaly detection system for vehicles. In *Smart Computing (SMARTCOMP), 2016 IEEE International Conference on*, pages 1–6. IEEE, 2016.

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

[37] 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.

[38] 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.

[39] O. Stan, Y. Elovici, A. Shabtai, G. Shugol, R. Tikochinski, and S. Kur. Protecting military avionics platforms from attacks on mil-std-1553 communication bus. *arXiv preprint arXiv:1707.05032*, 2017.

[40] I. Studnia, E. Alata, V. Nicomette, M. Kaâniche, and Y. Laarouchi. A language-based intrusion detection approach for automotive embedded networks. *International Journal of Embedded Systems*, 10(1):1–12, 2018.

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

[42] A. Theissler. Detecting known and unknown faults in automotive systems using ensemble-based anomaly detection. *Knowledge-Based Systems*, 123:163–173, 2017.

[43] D. Tian, Y. Li, Y. Wang, X. Duan, C. Wang, W. Wang, R. Hui, and P. Guo. An intrusion detection system based on machine learning for can-bus. In *International Conference on Industrial Networks and Intelligent Systems*, pages 285–294. Springer, 2017.

[44] 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.

[45] A. Wasicek and A. Weimerskirch. Recognizing manipulated electronic control units. Technical report, SAE Technical Paper, 2015.

[46] 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.

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

Pal-Stefan Murvay is an assistant professor at Politehnica University of Timisoara (UPT). He graduated his B.Sc and M.Sc studies in 2008 and 2010 respectively and received his Ph.D. degree in 2014, all from UPT. He has a 10-year background as a software developer in the automotive industry. His current research interests are in the area of automotive security and works as a postdoctoral researcher in the CSEAMAN project.