DAGA: Detecting Attacks to in-vehicle networks via n-Gram Analysis

Dario Stabili*, Luca Ferretti†, Mauro Andreolini† and Mirco Marchetti*
* Department of Engineering "Enzo Ferrari"
† Department of Physics, Informatics and Mathematics
University of Modena and Reggio Emilia, Italy.
Email: {dario.stabili, luca.ferretti, mauro.andreolini, mirco.marchetti}@unimore.it

***Abstract*—Recent research showcased several cyber-attacks against unmodified licensed vehicles, demonstrating the vulnerability of their internal networks. Many solutions have already been proposed by industry and academia, aiming to detect and prevent cyber-attacks targeting in-vehicle networks. The majority of these proposals borrow security algorithms and techniques from the classical ICT domain, and in many cases they do not consider the inherent limitations of legacy automotive protocols and resource-constrained microcontrollers. This paper proposes DAGA, an anomaly detection algorithm for in-vehicle networks exploiting $n-$gram analysis. DAGA only uses sequences of CAN message IDs for the definition of the $n-$grams used in the detection process, without requiring the content of the payload or other CAN message fields. The DAGA framework allows the creation of detection models characterized by different memory footprints, allowing their deployment on microcontrollers with different hardware constraints. Experimental results based on three prototype implementations of DAGA showcase the trade off between hardware requirements and detection performance. DAGA outperforms the state-of-the-art detectors on the most performing microcontrollers, and can execute with lower performance on simple microcontrollers that cannot support the vast majority of IDS approaches proposed in literature. As additional contributions, we publicly release the full dataset and our reference DAGA implementations.**

I. Introduction

In the the last years the automotive industry shifted towards the adoption of novel technologies such as drive-by-wire systems, Advanced Driving Assistance Systems (ADAS) and Internet connectivity, resulting in a proliferation of Electronic Control Units (ECUs) connected to heterogeneous sensors and actuators used to monitor and control the vehicle and its surroundings. Despite these systems are designed to increase safety and reduce the risk of road fatalities, the adoption of software-controlled actuations introduces security vulnerabilities [1] that are documented in white papers and technical reports [2], [3]. Moreover, the increasing adoption of Internet connectivity paved the way to novel attack vectors [4] that might expose sensitive information about the vehicle and its passengers. Since the first introduction of drive-by-wire systems in modern automobiles, security researchers from both industry and academia demonstrated that the vulnerabilities of these systems [5] might be hijacked to obtain remote control of the vehicle. The first countermeasures adopted by car manufacturers are borrowed from the ICT security domain: *secure gateway* and *authenticated diagnostic* are relevant examples of security solutions adapted to the automotive scenario [6], [7]. However, this first layer of defense fails against attackers that are able to access either the internal vehicle network or the ECUs. To this aim it is necessary to adopt a defense-in-depth paradigm for the whole vehicle system, and since internal vehicular communications are the main target of cyber-attacks to the connected vehicles [8] it is necessary to monitor the network communication continuously. To this purpose, many security researchers already proposed Intrusion Detection Systems (IDS) designed for the Controller Area Network (CAN) [9], which is the most deployed internal network communication protocol in modern vehicles. However, most of the existing research does not consider the computational constraints of the automotive microcontrollers, hence resulting in the proposal of intrusion detection models that are not deployable on common microcontrollers [10]. On the other hand, previous works that are focused on the proposal of solutions currently deployable on automotive microcontrollers are designed to identify only a limited set of attacks [11], [12]. In this work we present *DAGA (Detecting Attacks to in-vehicle networks via n-Gram Analysis)*, an anomaly detection framework designed to use $n-$grams for its detection purposes. The DAGA framework exploits the $n-$grams for the definition of different detection models that can be used to adapt the detection algorithm to the specifications of the microcontrollers. The detection capabilities of DAGA are tested against a threat model including 5 attack scenarios that are demonstrated effective in known attacks against modern vehicles. The experimental evaluation of DAGA demonstrates its ability to achieve high detection performance against all the different attack scenarios considered in the threat model. Moreover, three implementations of the DAGA detection algorithm are presented, showcasing the trade-off between the memory footprint of the detection timeliness on 5 microcontrollers used in automotive applications. As a final contribution, the dataset and the implementations of DAGA are publicly released.

The rest of the paper is organized as follows. Section II discusses related work. Section III presents the fundamentals required for the understanding of this paper and the considered threat model, while Section IV describes the dataset used

for training and testing DAGA against the considered threat model. Section V presents the DAGA detection algorithm and the design choices for three reference DAGA implementations. Section VI presents the experimental detection performance of DAGA against the considered threat model. The of DAGA for automotive-grade microcontrollers is presented in Section VII, showcasing the different trade-offs of the three reference implementations. Final remarks are discussed in Section VIII.

II. Related work

With the introduction of electronic components to in-vehicle networks, a modern car can be considered as a Cyber-Physical System (CPS). Cyber-security for CPS is an established field of research, focused on the development of detection methodologies for protecting the system and its components [13]–[15]. However, the peculiarities of the automotive domain hinder the adaptation of solutions designed for generic CPS to this scenario, requiring the design of novel solutions targeting its components [16]. Solutions currently investigated by security researchers are either focused on the development of novel security mechanisms targeting V2V communications [17] or in demonstrating the issues related to embedded devices [18] or to V2V communications [19].

Another active research path is focused on hardening in-vehicle networks by applying concepts borrowed from classical ICT networks to the automotive scenario. This results in the development of novel intrusion detection systems specifically designed for the CAN bus [20], which is the most deployed in-vehicle communication protocol. The baseline assumption for IDS is that it is possible to build a model that describes the normal behavior of the CAN bus, and that attacks can be detected because they introduce a measurable deviation from the normal profile [6]. Works within this field propose different models for the definition of the normal behavior of the CAN bus.

Security researchers demonstrated that anomaly detectors based on machine learning and neural networks can be adapted to the CAN bus [21], [22]. Despite early results looks promising, the application of machine learning detection algorithm or neural networks to the in-vehicle network does not comply with the strict requirements of automotive microcontrollers.

Due to the necessity of standardization of different security-oriented solutions, the AUTOSAR consortium [23] is established since 2003. AUTOSAR provides a set of specifications that describe basic software modules, application interfaces, and define a standard exchange format that manufacturers and suppliers can adopt for their needs. AUTOSAR covers functional safety and security aspects of on-board communications in the *Secure On Board Communication* basic software module (SecOC), listing its requirements [24] and providing its specification [25]. Many solutions designed to extend the AUTOSAR SecOC module have already been presented in literature [26], [27] and proven efficient in protecting on-board communications. All the AUTOSAR solutions are focused on the authenticity, confidentiality, and non-repudiation of the CAN communications, while the solution presented in this paper is focused on the proposal of an anomaly detector for CAN communications. Since the AUTOSAR SecOC solutions and the one presented in this paper are focused on different aspects, it is possible to deploy them independently from each others to further increase the security of CAN communications.

Many intrusion detection systems for in-vehicle networks have already been published in literature. Some of these intrusion detection systems are designed to analyze the low-level characteristics of the ECUs composing the internal vehicle network, such as the voltage differentials of CAN transceivers [28], [29]. These solutions are able to detect any inconsistency by comparing the low-level characteristic evaluated during transmission of a message against the detection model. However, compared to the other detection methods presented in literature, these methods require a dedicated hardware for their implementation, preventing their deployment on common microcontrollers. Moreover, similar approaches are only effective against an attacker that replaces or impersonates a benign ECU with different hardware, while it cannot detect attacks based on the exploitation of a software vulnerability of a benign ECU.

Other research efforts are focused on the analysis of the inter-arrival times of the CAN messages and on the analysis of the content of those messages. Anomaly detectors based on the analysis of the inter-arrival times [11], [30], [31] are based on the assumption that most CAN messages are sent periodically on the network within a fixed time interval, hence it is possible to exploit this feature to detect messages that do not follow the expected timing. These detection methods are only applicable to cyclic messages and cannot detect any anomaly if the attack targets a non cyclic message. The experimental results of [31] demonstrated a false positive rate of 0.294%, which is extremely high in the automotive scenario since high speed CAN bus can deliver thousands of messages per second. Moreover, the detection method described in [31] is based on the signals encoded in the payload CAN data frames, hence requiring the access to the formal specification of the vehicle or the application of reverse engineering methods to extract signals from CAN data [32], [33]. Our proposal, that does not analyze the contents of the payload, shows that in the context of CAN communications even lightweight analyses only based on CAN IDs can be effective to detect many classes of attacks.

Another group of solutions is based on the definition of thresholds applied to high level features computed by aggregating multiple CAN messages [34], [35]. The authors of [34] presented a detection method based on the analysis of the entropy of the network, using a threshold-based metric for the definition of the normal entropy values, while the authors of [35] proposed a threshold-based detection model based on the spectrum analysis of chunks of CAN message payloads. All these solutions require floating point operations, which are usually not supported by low-end microcontrollers. Moreover, these methods are only effective against high-volume attacks, in which the attacker injects hundreds of malicious messages per second, making them unreliable against low-volume and targeted attacks.

Other works presented detection models based on the content of CAN messages. Some of these works are based on the analysis of the content of the message payload. These solutions

either (i) require access to the specification of the vehicle to extract the signals encoded in the payload [36], (ii) have been designed to detect only a particular attack scenario [12], or (iii) leverage detection algorithms that do not meet the computational constraints of common microcontrollers [10], [36]. Lightweight algorithms [37], [38] that do not rely on proprietary formal specifications and are deployable on low-end microcontrollers are based on the analysis of message identifiers. However their low computational cost are characterized by poor detection performance against many known attacks. Similarly to the aforementioned work, DAGA relies on message identifiers for its detection purposes by exploiting $n-$gram to build its detection model. The $n-$grams used by DAGA are composed of a sequence of n consecutive identifiers transmitted over the CAN bus. Despite $n-$gram analysis has been applied at first for natural language processing [39], [40], it has been also used for the development of IDS in ICT networks [41], [42] . To the best of our knowledge, DAGA is the first $n-$gram based IDS applied to the automotive scenario, and the extensive experimental evaluation presented in this paper shows that DAGA is able to achieve excellent detection performance against a wide range of real attack scenarios. We remark that the aim of DAGA is to demonstrate the suitability of $n-$gram-based detection to resource-constraint devices such as the ones characterizing the automotive environment. The current state-of-the-art includes many anomaly detectors based on detection metrics designed to operate on the different fields of the CAN frames. There are at least 3 main benefits on focusing on the ID of the CAN message for detection purposes. The first one is that it is possible to access the content of the ID field without any reverse engineering process since the message IDs are defined in the CAN standard. Second, the injection of CAN messages is the final goal of any known attack to the CAN bus (Section III-B), hence it is possible to identify a wide array of different attacks by monitoring only the sequence of message IDs on the network. Finally, the experimental evaluation on real automotive microcontrollers demonstrates that by using detection models with different memory footprint it is possible to deploy DAGA on extremely resource-constraint devices, including the vast majority of ECUs found inside a modern vehicle. Compared to the state-of-the-art, DAGA does not require a dedicated hardware, it is effective also in the presence of non periodic CAN messages, it is able to detect low-volume attacks, and it does not require access to proprietary vehicle specifications. Moreover, to the best of our knowledge DAGA is the first anomaly detection framework that offers the possibility to tailor the detection method according to the desired detection performance, the characteristics of the target microcontroller, and the desired detection timeliness.

III. Background knowledge and threat model

We describe the background knowledge required for the understanding of the paper in Section III-A and the threat model Section III-B.

A. A primer on CAN

The Controller Area Network is a vehicle bus standard designed to allow the nodes of the network to exchange data without requiring a host computer [9]. CAN is one of the most deployed networking protocols for internal vehicular communications due to its high resilience to electromagnetic interference and its cheap implementation.

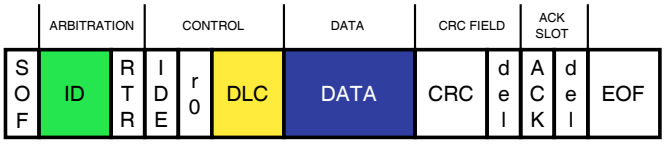

Fig. 1: Data frame standard format

The CAN protocol defines 4 different message types. Among these 4 message types, the *data frame* is the only one used to exchange data between ECUs. The CAN data frame is composed of 3 main fields: the *identifier (ID)*, the *data length code (DLC)* and the *payload (data)*. The *ID* is used to distinguish among different types of CAN data frame. Data frames characterized by a given ID are produced by only one ECU, while receiver ECUs use the value of the ID to select data frames that are relevant for their functioning. The ID is also used for arbitration of the CAN messages, where lower values of this field denote messages with higher priority. The size of the ID field depends from the format of CAN message. Figure 1 shows an example of a CAN data frame in the *standard* format, which have IDs with a size of 11 bits, while the *extended* format defines the ID field with a size of 29 bits. The extra 18 bits of the *extended* format are encoded separately from the 11 bits of the *standard* format for backward compatibility. The *DLC* field encodes the number of bytes composing the *data* field. The *data* field encapsulates the information that the sender ECU transmits to other ECUs on the network. The data field has a variable size (from 0 to 8 bytes) and usually packs several different signals. The CAN standard leaves complete freedom to the car manufacturers about the structure, number, encoding, and semantic of these signals. Hence, without having access to the formal specifications of the CAN messages for a particular vehicle model (contained in a file called DataBase for CAN or DBC), the signals encoded in the data field can only be interpreted as an opaque binary blob.

B. Threat model

We consider attackers that have access to the CAN bus and are able to: (I) sniff the CAN data and learn information about the normal CAN traffic flow; (II) inject arbitrary CAN data frames without any limitation about their content or the injection frequency. We also assume that attackers are *not* able to manipulate, re-order or delete messages in transit on the CAN bus. We observe that DAGA should not be used in scenarios where attackers can completely control the medium (e.g., MitM attacks). A similar threat model allows the attacker to perform multiple realistic attacks already considered by security researchers, including *replay attacks* [2], [8], *fuzzing attacks* [43], and *denial-of-service attacks* [44], [45]. We

remark that all the aforementioned works are focused on gaining access to the CAN bus to inject messages. How the infection of the CPS system is achieved can differ from work to work, but the final goal is to inject messages on the CAN bus. In [2] the authors accurately described the process of gaining remote access to the infotainment unit of their target vehicle through the remote exploitation of exposed vulnerable services, which led to the re-flashing of the gateway ECU, allowing the injection of messages from the infotainment unit to the CAN bus, while in [45] the authors physically connected a malicious device to the CAN bus to deactivate an ECU by exploiting the CAN error handling mechanism. While attacks can be extremely different from each other, they both require the injection of values on the CAN bus to affect the physical behavior of the target vehicle. Hence, protecting the bus from these scenarios is crucial for the safety of the automotive CPS. The details of the different attacks composing our threat model are described in the following.

1) Replay attack: A replay attack is performed by gathering legit messages from the CAN bus and injecting them at a later time. The aim of this attack is to subvert the normal behavior of the ECUs, including those powering drive-by-wires systems [8], [46]. We highlight that in this attack model the attacker cannot remove normal messages from the bus, hence replayed messages interleave with the normal CAN traffic.

As an example, consider the scenario in which a collision avoidance system uses the front sensors to detect the presence of objects in front of the vehicle and that the related CAN messages is sent with a cycle time of 10ms. The attacker might replay these message with a higher frequency, e.g. a message every millisecond. Moreover, by reverse engineering the content of the message (via a manual reverse engineering analysis or using automatic reverse engineering tools such as [32], [33]), the attacker is able to craft the content of the injected message. As an example, the attacker can record a CAN data frame with the meaning of *"front road clear"*, then it can change the content of the message with the same value used to represent the presence of obstacles (i.e. *"front obstacle - brake"*), thus activating the collision avoidance system.

We consider three different types of replay attacks:

- **Single message replay:** a single message is gathered from the bus and then injected at a later time;
- **Ordered sequence replay:** an ordered sequence of messages is gathered from the bus and then injected at a later time without any modification;
- **Arbitrary sequence replay:** after gathering a set of messages from the bus, the attacker rearranges them to generate an arbitrary sequence of messages, which is then injected at a later time.

2) Fuzzing attack: Fuzzing is the process of submitting malformed inputs to a system aiming at eliciting anomalous behaviors that might expose unknown vulnerabilities and might help in reverse-engineering the syntax and semantic of undocumented communication protocols [47]. Fuzzing attacks in automotive networks are implemented by injecting CAN messages with random content. Current literature describes two types of fuzzing attacks:

- **ID fuzzing:** the attacker generates malicious CAN data frames having random ID and data, and injects them into the bus. The ID field is chosen to be different from any ID observed in the normal traffic;
- **Payload fuzzing:** the attacker generates malicious CAN data frames having a valid ID (i.e. an ID that has been previously seen on the bus) and a randomly generated data field, and injects them into the bus.

Let us refer to the previous example of the front sensor used for collision avoidance. An attacker can fuzz the payload of this message to reverse engineer its semantic.

3) Denial-of-Service: The Denial-of-Service (DoS) attack aims to disrupt a normal process by preventing legitimate access to a necessary resource for a significant time interval [48]. In automotive networks, DoS attacks can be performed by injecting high-priority messages at a high-frequency rate, preventing legit lower-priority messages from being transmitted (see Section III-A). We consider a DoS attack to be effective only if it disrupts the normal CAN communication for at least twice the cycle time of the most frequent message on the network. As an example, if the most frequent periodic message has a cycle time of 10ms, the DoS attack is considered effective only if it last for at least 20ms. We consider two types of DoS attacks strategies:

- **Zero ID DoS:** the attacker injects CAN data frames with the ID field set to $0x000$. Although these messages would always win arbitration, simple detection strategies can easily detect the attack in case the ID $0x000$ is not used by legit messages;
- **Lowest ID DoS:** the attacker injects CAN data frames with the ID field set to the lowest ID value previously observed among legit messages.

Consequences of this attack have been studied in literature [49]. While non-critical ECUs stop working in case of missing CAN communications, critical ECUs (such as the Engine Control Unit or the Brake Control Module) are designed to function in limited regime (also known as *limp mode*) to prevent mechanical damages to the vehicle. In limp mode, critical ECUs continue to function without participating in the normal CAN communication, allowing the driver to safely stop the vehicle. The limited working regime affects the subsystems of the vehicle differently according to their scope. As an example, upon activation of the *limp mode* the driving subsystem may limit the vehicle's speed under 30 km/h while also limiting the gearbox to the second gear. If a DoS attack is conducted while the target vehicle is running on a high-speed road (such as highways or motorways) the limited working regime of critical ECUs might be used to jeopardize the safety of people inside and nearby the vehicle. For completeness, we remark that limp mode has also been proposed as a reaction mechanism following the detection of cyber-attacks to in-vehicle networks [50].

IV. Dataset

This section describes the dataset used for training and validating DAGA. The dataset is collected from the CAN bus of an unmodified, licensed 2016 Volvo V40 Kinetic model.

The CAN communication is recorded by physically connecting a laptop to the On-Board Diagnostic (OBD-II) port with a PCAN-USB adapter by Peak System [51] and a D-Sub to OBD-II cable. The high-speed CAN bus segment exposed on the OBD-II port of the vehicle contains data coming from the *powertrain* segment, hence it is possible to access to CAN data frames exchanged by the ECUs to control the engine and other systems found in this section. The dataset presented in this section is composed of two different parts: the *clean* part, used for the training of DAGA and containing CAN traffic traces recorded from the test vehicle; and the *infected* part, used for the performance evaluation of the detection of DAGA and in which the attack composing the threat model described in Section III-B are replicated. To the best of our knowledge, this is the first dataset containing multiple attacks designed for the evaluation of CAN IDS, and is publicly available online [1].

A. Clean dataset

The clean dataset is composed of 7 different CAN traces, including more than 8 million CAN messages corresponding to approximately 90 minutes of CAN traffic. The CAN traces are gathered in different driving sessions performed on different road types (urban, suburban, and highway), traffic conditions, weather conditions and geographical areas (plain, hill, and mountain), and by activating many different control commands. The CAN traces include ID, DLC, and payloads of each CAN data frame associated to its relative timestamp. The clean dataset contains data from the *powertrain* segment of the CAN bus of our test vehicle, which is used by ECUs to exchange data related to the vehicle dynamics. The other CAN segments are usually identified as the *body* (used to control different comfort features such as the windshield wipers and the air conditioning), the *chassis* (used for the centralized lock or the output of the proximity sensors), and the *infotainment* segments (used to control the radio and the external connections exposed by the vehicle). These CAN segments are connected with each other through a centralized gateway ECU, and despite the segments are usually isolated from each other, some CAN messages might be forwarded from one segment to the other to implement particular features, such as increasing the stereo volume in case the engine RPMs are above a defined threshold. The monitored segment of the CAN of our test vehicle exposes 50 unique message IDs, which are all available in all the 7 traces. Despite this number seems little compared to the overall number of possible CAN IDs (2^{11} on a standard CAN implementation), we remark that the *powertrain* section of the vehicle is only one of the different CAN segments found in modern vehicles.

As an example, consider that trace #1 is recorded on a sunny day while trace #3 is recorded in a rainy day, activating both front and back windshield wipers. Hence, it is logical to assume that trace #3 should include the data related to the activation of the wipers. However, by comparing the IDs found in trace #3 with the ones found in trace #1 no difference is detected. This example shows two interesting behaviors. The first one is that messages whose content is not required by ECUs in the *powertrain* segment of the network are not forwarded into the segment, hence limiting the overall number of message IDs found in the segment. The second interesting behavior is that the CAN message IDs found in each of the 7 traces is consistent, despite the activation of different control commands. This implies that despite the overall number of possible CAN IDs is 2^{11}, only 50 of those messages are actually used in the *powertrain* segment of our test vehicle.

[1] https://github.com/SECloudUNIMORE/ACS/tree/master/DAGA

B. Infected dataset

The *infected dataset* is used for the evaluation of the detection performance of DAGA, and is generated by simulating the attacks composing the threat model presented in Section III-B on the traces of the clean dataset in a laboratory environment for safety reasons. The laboratory environment used for the generation of the attack traces is composed of a laptop computer, a Raspberry Pi 4 board, and an Arduino Mega. The CAN bus is implemented through a breadboard, and each device is directly connected to the bus. The expansion boards used to connect both Raspberry and Arduino to the CAN are terminated with a 120Ω resistor and include a CAN transceiver responsible to handle re-transmissions, delays, arbitration and, in general, all the low level details that might have been affected by simulation artifacts. The Raspberry board is used to replay the clean CAN trace, while the Arduino board is used to inject malicious CAN messages. The laptop computer records the data from the CAN bus that is used in the experimental evaluation. Figure 2 shows the laboratory environment used for the generation of the infected traces.

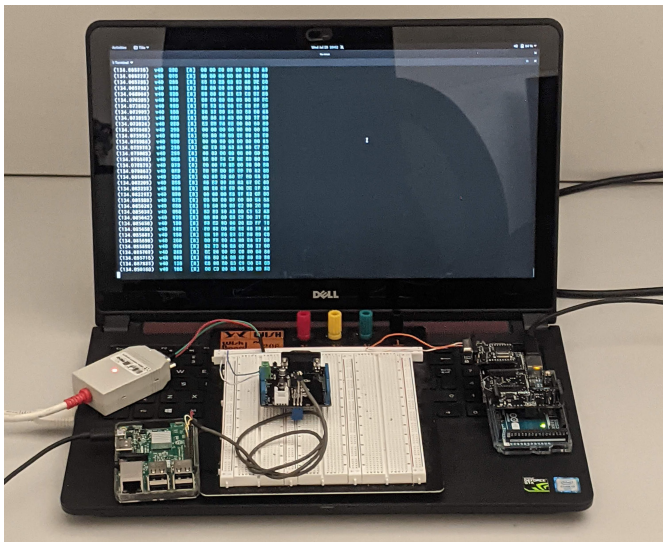

Fig. 2: Laboratory setup used for the generation of the infected dataset

To generate the different attack scenarios it is necessary to consider the messages injected by the attacker. As an example, consider a low-volume replay attack targeting a message that normally appears on the CAN with a very high frequency. This attack might go undetected with a higher probability with respect to the same attack targeting a message that normally appears on the bus with a very low frequency or with a non-periodic pattern. To overcome this limitation and perform a

comprehensive performance evaluation, the infected dataset includes multiple instances of the different attack scenarios, each one targeting a different set of messages that are selected based on their different probabilities of appearing on the CAN bus. As a result, the infected dataset includes a total of 168 CAN traces, for a total of more than 200 million messages corresponding to more than 40 hours of CAN traffic. All the injection attacks are generated with a fixed frequency of one attack instance every second. This attack frequency is selected as representative of low-volume attacks, which are the most difficult to detect for any IDS designed for CAN communications. Hence, we remark that the selected frequency is also the worst-case scenario for the detection performance of DAGA.

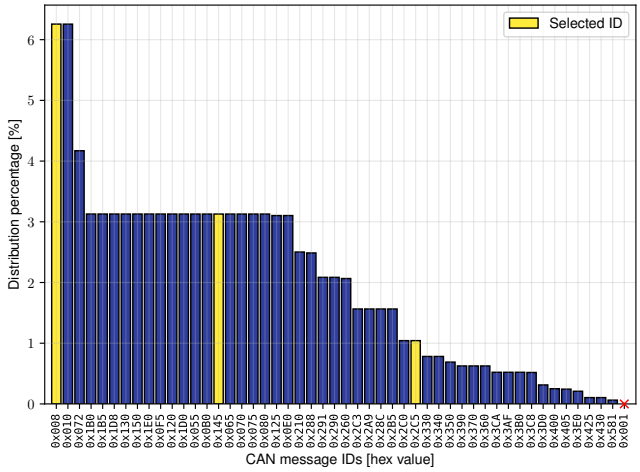

Fig. 3: Percent distribution of the IDs composing the dataset

Figure 3 shows the probability distribution of the messages of the clean dataset. The x-axis represents the message IDs, while the y-axis shows the percentage of those messages in the clean dataset. The 3 bars highlighted with a different color represent 3 cyclic message IDs selected for the simulation of the different attack scenarios. They are selected as representative of the messages with a high, medium, and low frequency (IDs `0x008`, `0x145`, and `0x2C5`, respectively). We also included the only non-periodic message found in the dataset (ID `0x001`) to better represent the different class of messages. The percent distribution, the cycle-time, and the label used to address the selected messages are summarized in Table I.

ID [hex]	Distribution [%]	Cycle-Time [ms]	Label
0x008	6.2565	10	`top`
0x145	3.1278	20	`mid`
0x2C5	1.0426	60	`low`
0x001	8.0057e-5	-	`not`

TABLE I: IDs of messages selected from the clean dataset

The infected dataset is composed of the following attacks:

- **Single ID replay:** set of traces in which a single message with a valid ID is injected once every second. This attack scenario is composed of 4 different attack instances, each corresponding to the injection of one of the 4 selected messages over the 7 traces of the clean dataset, for a total of 28 traces. We remark that this attack scenario can be further distinguished depending on the content of the payload field. In case of a valid payload, obtained through observation of messages included within the clean dataset, the attack is declined as *single message replay*; otherwise, in case of a payload chosen through other methodologies, the attack is declined as *payload fuzzing*. Although for other existing approaches these two attacks have different consequences on the detection system (and can be detected with other detection algorithms), in our proposal they both fall back to the same class of attack because both of them require the injection of a single message with a valid ID. Hence, since our solution is not able to distinguish between the two attack scenarios, the detection performance of DAGA is tested against the generic *single ID replay* attack scenario.
- **Ordered sequence replay:** set of traces in which sequences of messages observed in the clean traces are injected once every second. Injected sequences are randomly selected from the clean dataset and their length varies from a minimum of 2 to a maximum of 10 messages, thus generating 9 different attack instances for each of the 7 clean traces. Hence this scenario includes 63 traces.
- **Arbitrary sequence replay:** set of traces in which a randomly generated sequence of valid messages is injected once every second. The length of injected sequences ranges from a minimum of 2 to a maximum of 10 messages. Also this scenario includes 63 traces.
- **ID fuzzing:** set of traces in which a message having a random ID that does not belong to the clean dataset is injected once every second. The payload of this message is also randomly generated, and the total number of infected traces generated in this scenario is composed of 7 traces.
- **Lowest ID DoS:** set of traces in which a Denial-of-Service attack is simulated by injecting the message with the lowest ID. The most frequent message of the clean dataset has a cycle time of 10ms. According to the assumption presented in Section III-B3, the DoS attack is considered effective if the normal communication is disrupted for at least 20ms. Since the CAN segment of our vehicle is configured with a baud rate of 500kbps (which is the most common baud rate for high-speed CAN bus of modern vehicles), it is necessary to inject at least 10k bits to ensure that the bus will be filled with DoS messages for the required time. By injecting CAN data frames with a length of 111 bits, to occupy the bus for a 20ms it is necessary to inject 90 messages, resulting in an injection frequency of 4500 messages each second. The injected messages have a fixed ID value equal to the lowest ID found in our dataset ($0x1$). This attack scenario is composed of 7 traces. The *zero ID DoS attack* is not simulated since the ID $0x0$ is not found in the clean dataset, hence the detection performance against this attack are the same of the ones achieved against the ID fuzzing attack.

V. THE DAGA FRAMEWORK

This section introduces the DAGA framework. The DAGA detection algorithm is described in Section V-A, while the reference implementations of DAGA used for the feasibility analysis on the automotive microcontrollers are presented in Section V-B.

A. Description of the detection algorithm

The DAGA detection algorithm exploits $n-$gram analysis for detecting anomalies in CAN communications. Each $n-$gram used for anomaly detection is composed of the ordered sequence of CAN message IDs. DAGA only relies on the CAN message IDs for its detection purposes, without requiring access to any other field of the CAN data frame. By using the DAGA framework it is possible to define multiple detection models, each one characterized by a different value of the parameter n. Intuitively, we can expect models with a higher value of n to better represent the normal CAN traffic, and to achieve better detection performance. However, increasing the value of the parameter n also increases the size of the model and the memory requirements to execute an instance of DAGA. To prevent the inapplicability of DAGA to automotive microcontrollers due to memory constraints, the parameter n is kept as a configurable parameter, allowing the analysis of the trade-off between the memory requirements and the detection performance. Since the value of the parameter n does not affect the definition of the algorithm, the detection algorithm is described generically.

The detection algorithm is composed of two phases. In the first phase the clean dataset is used for training the detection model according to the desired value of the parameter n. In this phase, the algorithm extracts the $n-$grams from the traffic traces as sequences of n message IDs in a sliding window fashion. We highlight that DAGA does not differentiate between "popular" and "unpopular" $n-$grams, hence their frequency is not considered. The detection model is composed of all the unique $n-$grams that are found in this phase, discarding all duplicates. The pseudocode for the training phase of the algorithm is shown in Algorithm 1.

Algorithm 1 Routine for training DAGA with a given value of the parameter n

```
model ← null
for trace in dataset do
    ixe ← n
    while ixe ≤ trace.length do
        seq ← trace[ixe − n : ixe].CAN_ID
        if seq ∉ model then
            model.add(seq)
        end if
        ixe + +
    end while
end for
save(model, n)
```

In the detection phase the algorithm evaluates the $n-$grams from the monitored CAN bus by extracting sequences of message IDs of length n, with n equal to the same value used in the training phase. The $n-$grams extracted from the CAN communication are compared with the $n-$grams composing the detection model. In case the extracted $n-$gram is not found in the detection model, then an anomaly is raised. The detection phase continues in a sliding-window fashion, hence creating the next $n-$gram by removing the first ID from the current $n-$gram and appending the next ID to it.

The pseudocode for the detection phase of the algorithm is shown in Algorithm 2.

Algorithm 2 Routine for $n-$gram membership test used in the detection phase of DAGA

```
mdl, n ← load(model)
seq ← null
for msg in stream.curr_msg do
    seq.add_tail(msg.ID)
    if seq.length == n then
        if not ValidNGram(seq, mdl) then
            raise anomaly
        end if
        seq.pop_head()
    end if
end for
function ValidNGram(sequence, model)
    if sequence ∈ model then
        return True
    end if
    return False
end function
```

B. Design of DAGA implementations

In this section we present three different prototype implementations of DAGA. These implementations are designed to offer a comparable trade-off in terms of computational and memory requirements. At first, we describe a pre-processing step that is used by all implementations. Then, we present the three designs: an implementation based on the binary lookup algorithm used as a baseline for the comparison of the other two implementations, a lookup-optimized implementation based on hash tables and a memory-optimized implementation based on an original data structure similar to the *sparse trie* data structure. A comparison of the performance of the three implementations deployed on automotive-like microcontrollers is presented later in Section VII. The different implementations of DAGA are publicly available online [2].

Symbols notations. We denote the number of unique IDs within the dataset as u, the size of each ID as ID_{len}, the number of $n-$grams within the dataset used for detection as $size$, and the length of each $n-$gram as n.

Dataset encoding. The baseline size of a model is $size \times n \times ID_{len}$ bits. The value of ID_{len} equals to 11 bits if the messages are in the standard format, or 29 bits if the messages are in the extended format. We propose to encode the ID values

[2]https://github.com/SECloudUNIMORE/ACS/tree/master/DAGA

as a sequence of progressive integer numbers, which reduces memory usage of the dataset to $size \times n \times \lceil \log_2 u \rceil$ bits, plus a lookup table of size $u \times (ID_{len} + \lceil \log_2 u \rceil)$ bits. Detection requires to first convert each ID by using the lookup table, then to test membership of the $n-$gram within the dataset.

Baseline implementation: binary lookup. The binary lookup implementation of DAGA is implemented in the `C` programming language. The detection model is implemented with an array data structure, whose elements are the $n-$grams composing the detection model. Since the message IDs are encoded as a progressive number, it is possible to store all the different values in a 8-bit unsigned integer (`uint8_t`), hence the overall memory usage of the binary lookup detection model is $size \times n$ bytes. The $n-$grams of the detection model are sorted in a lexicographic ascending order to enable a binary lookup on the elements of the detection model. Our implementation of the binary lookup algorithm follows the one available on the NIST Dictionary of Algorithms and Data Structures [52], adapting the comparison to the $n-$gram data structure. The computational complexity for the lookup operation is equal to $\mathcal{O}(\log size)$.

Lookup-optimized implementation: hash table. The hash table implementation of DAGA is implemented in the `C++` programming language using the `std::unordered_set` container, characterized by fast lookup operations on its elements. Hash tables maintain data within an array of buckets, and the index of the bucket in which each value is stored is evaluated with a hash function on the value. Membership test is operated by re-computing the hash function and by operating a linear search over the bucket. Although memory usage seems comparable to the one of the binary lookup implementation, we observe that an hash table might include hidden overheads due to unused memory within buckets, which in turn depend on the quality of the hash function. We remark that the details about the the memory required for the `std::unordered_set` container and the implementation of the search function using this data structure (which is implemented through the `find` method) are highlighted in the official `C++` documentation [53]. The hash table implementation of DAGA has a computational complexity for the lookup operation of $\mathcal{O}(1)$.

Memory-optimized implementation: sparse trie. As a memory-optimized implementation we design a search tree, which we denote as *sparse trie*, that can be modeled as a *trie* where each node includes an encoded CAN ID, and whose size depends on the actual number of values stored in the trie. The sparse trie is designed to reduce memory usage with regard to the following characteristics:

- the height of the trie is the *sequence size* of the anomaly detection model;
- the maximum fan-out of each internal node is the number of CAN IDs, which is 50 for the considered dataset;
- with the exception of the lowest depths, the actual fan-out of the trie is very small, which is between 1 and 2 on average for depths higher than 2;
- the trie is built once during the training phase of the model and must only support read operations.

The sparse trie uses approaches related to efficient storage management, including byte-unaligned data, delta encoding and run length compression [54]. Moreover, it uses superimposed indexing and skip values to achieve practical lookup timings [54]. Although each of the adopted techniques is not novel, to the best of our knowledge the design of the sparse trie as a whole can be considered a special-purpose data structure. In the following, we overview the design of the sparse trie and we describe the parameters used to instantiate it within the considered workload.

Sparse trie design. The sparse trie includes three types of data, that we denote as *sparse values*, *indexes* and *skips*. *Sparse values* are a list of arrays, where each array includes the traversals of all the values stored at a certain depth level of the trie from left to right in a sparse fashion, that is, without including empty nodes. Each sparse value array stores CAN IDs by using a fixed-size representation in a byte-unaligned fashion, where the size of each value is the ceil of the binary logarithm of the number of symbols that must be represented. *Indexes* include information to point at the different *sparse values* nodes for lookup operations. For space efficiency, we maintain them in a compressed fashion by representing positions as *delta-encoded* values and by compressing them with *run length compression*. We discuss sizes and byte-alignment strategies of indexes below, when we compute the specific parameters used for the used dataset. *Skips* are aggregated sums of indexes values, and allow to improve the efficiency of lookup on *sparse values* and on *indexes*. Although theoretically they are not mandatory to build the data structure, they are necessary to achieve acceptable lookup speeds.

We overview the algorithms used to build the sparse trie and to lookup a sequence. For the sake of clearness, we do not consider byte-unaligned data and run length compression strategies.

Algorithm 3 shows how to build indexes and sparse values by using an existing trie data structure (Line 4). We assume existence of the $trie.traverse\_children(depth)$ routine of the trie to retrieve a list where each element is the set of all the non-empty children of a node at depth $depth$ (Line 7). Then, the list is *flattened* to be stored as an array within the sparse values list (Line 12). Delta-encoded indexes are built by storing all the sizes of children sets (Line 9).

Algorithm 4 shows how to build skips. For simplicity, the proposed function takes as input indexes and sparse values arrays associated to a depth of the trie, and can be used as-is with all the due depths. The function is quite simple as it only provides a way to aggregate sums of multiple delta-encoded indexes. However, we observe that an important design choice is the size of skips, that we design as the square root of the size of sparse values to optimize lookup time (Line 6). Similar sizing approaches are typical of other superimposed indexing structures, such as those used in vEB trees [55].

Algorithm 5 shows the lookup operation for verifying membership of a sequence in the sparse trie. The operation is an iterative procedure that descends from the root to the leafs of the sparse trie for each matched symbol. For each iteration, the procedure includes three parts: usage of skips (Lines 7-

Algorithm 3 Routine for building the sparse trie structure

```
1: function BUILD_SPARSE_TRIE(model, n)
2:     indexes ← list()
3:     sparse_values ← list()
4:     trie = Trie(model)
5:     for depth = 1 to (n − 1) do
6:         indexes_i ← list()
7:         nodes_list ← trie.traverse_children(depth)
8:         for nodes in nodes_list do
9:             indexes_i.append(len(nodes))
10:        end for
11:        indexes.append(indexes_i)
12:        sparse_values.append(flattenize(nodes_list))
13:    end for
14:    return (indexes, sparse_values)
15: end function
```

Algorithm 4 Routine for building skip data structures for a single layer of the sparse trie

```
1: function BUILD_SKIPS(indexes, sparse_values)
2:     skip_indexes ← list()
3:     skip_values ← list()
4:     sum_indexes ← 0
5:     sum_values ← 0
6:     skip_size ← ⌊√len(sparse_values)⌋
7:     for i = 0 to (len(indexes) − 1) do
8:         next_size ← (sum_indexes + indexes[i])
9:         if next_size > skip_size then
10:            skip_indexes.append(sum_indexes)
11:            skip_values.append(sum_values)
12:            sum_indexes ← indexes[i]
13:            sum_values ← sparse_values[i]
14:        else
15:            sum_indexes ← next_size
16:            sum_values ← (sum_values + sparse_values[i])
17:        end if
18:    end for
19:    if sum_indexes > 0 then
20:        skip_indexes.append(sum_indexes)
21:        skip_values.append(sum_values)
22:    end if
23:    return skip_size, skip_indexes, skip_values
24: end function
```

14), usage of indexes (Lines 14- 18), search over sparse values (Lines 18- 28). The worst case on asymptotic memory cost and asymptotic average computational complexity are $\mathcal{O}\left(size \cdot \log(u)\right)$ and $\mathcal{O}\left(\sqrt{size}\right)$, respectively. We remark that these worst cases are evaluated by considering a scheme that does not use compression techniques. As typical for data structures that use compression techniques, the performance of the proposed scheme vary depending on the characteristics of the data [54]. Thus, we evaluate the experimental performance of the sparse trie with the considered dataset in Section VII.

Parameters of compression strategies. We denote as *symbol size* the number of bits used to store symbols, and as *rle size* the number of bits used to store counters in run-length compression. The *symbol size* value that allows to minimize memory usage can be computed as $\lceil \log_2(s) \rceil$. Since the number of unique CAN IDs in the considered dataset is 50, the *rle size* value in our implementation is equal to 6. The best *rle size* value can be computed experimentally to optimize the efficiency of *run-length encoding*. In our design, compressed indexes are stored as couples of values $(rle\_counter, symbol)$. Thus, the value of *rle size* might affect

Algorithm 5 Routine for sparse trie sequence membership verification

```
1: function VERIFY_STREE(seq, n, indexes, sparse_values, skips)
2:     skip_sizes ← skips.skip_sizes
3:     skip_indexes ← skips.skip_indexes
4:     skip_values ← skips.skip_values
5:     i_seek ← seq[0]
6:     for i = 0 to (n − 1) do
7:         i_skip ← 0
8:         v_seek ← 0
9:         quotient, residuous ← divmod(i_seek, skip_sizes[i])
10:        i_seek ← residuous
11:        for j = 0 to (quotient · skip_sizes[i] − 1) do
12:            i_skip ← (i_skip + skip_indexes[i][j])
13:            v_seek ← (v_seek + skip_values[i][j])
14:        end for
15:        for j = i_skip to (i_skip + i_seek) do
16:            v_seek ← (v_seek + indexes[i][j])
17:        end for
18:        node_size ← indexes[i][i_skip + i_seek + 1]
19:        found ← False
20:        for j = 0 to (node_size − 1) do
21:            if sparse_values[i][v_seek + j] = seq[i + 1] then
22:                found ← True
23:                break
24:            end if
25:        end for
26:        if found ≠ True then
27:            return False
28:        end if
29:        i_seek ← (v_seek + j)
30:    end for
31:    return True
32: end function
```

the alignment of the *rle counter* values, which is *symbol size* plus *rle size*, and thus the speed of lookup operations. Experimental evaluations showed that the best *rle size* value to optimize memory usage with the considered dataset would be 3. However, we decide to set it to 2 because in our experimental setting, where *symbol size* is equal to 6, it allows to build byte-aligned packets that allows faster lookup times. Memory usage is slightly higher, but it does not prevent usage of the scheme on any platform, and significantly improves lookup times.

VI. Experimental evaluation

The detection performance of DAGA are evaluated by means of $\mathcal{F}-$measure, which is a statistical index representing the accuracy of a test. The $\mathcal{F}-$measure is evaluated using the *precision* (i.e. the number of correctly identified anomalies on the total of detected anomalies, hence including also the false positives) and the *recall* (i.e. the number of correctly identified anomalies on the total of actual anomalies, hence including the false negatives). Being tp, fp, and fn the number of true positives (i.e. non-legit instances correctly identified as anomalies), false positives (i.e. legit instances erroneously detect as anomalies), and false negatives (i.e. non-legit instances erroneously identified as legit), the *precision* is evaluated as $\frac{tp}{tp+fp}$, the *recall* is evaluated as $\frac{tp}{tp+fn}$, while the $\mathcal{F}-$measure is evaluated as $2 * \frac{precision*recall}{precision+recall}$. The $\mathcal{F}-$measures index ranges in the interval $[0, 1]$, where values close to 0 denotes low detection capabilities and values close 1 denotes high detection capabilities.

The DAGA detection algorithm is tested against the infected dataset presented in Section IV. Different values of the parameter n (from 1 to 10) are used to train different detection models to compare their results. Higher values of n are not tested since it is possible to achieve a $\mathcal{F}-$measure ≥ 0.99 with values of $n = 10$, while further increments of n do not lead to an significant increment of the $\mathcal{F}-$measure.

A. Single ID replay

Figure 4 shows the detection performance of DAGA against the single ID replay attack. The y-axis of Figure 4 represents the $\mathcal{F}-$measure evaluated using models created with a particular value of the parameter n, depicted on the x-axis. The four lines of Figure 4 represent the different message IDs used for the attack generation (see Table I). In particular, the green line represents the results achieved against the injection of the `top` message, the cyan line represents the results against the injection of the `mid` message, while the orange and the red lines represent the results against the injection of the `low` and `not` messages, respectively. The results are presented by means of box-plot to highlight the variance against the different simulated scenarios, while solid lines connect the median $\mathcal{F}-$measure of each set of experiments to highlight the trend for growing values of n. For readability purposes the outliers of the box-plots are omitted.

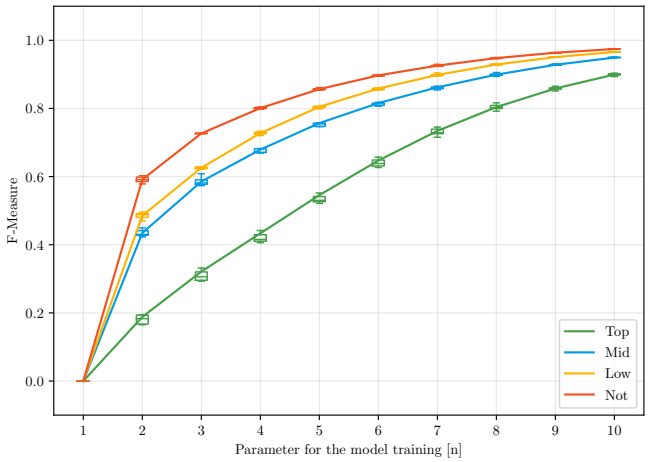

Fig. 4: $\mathcal{F}-$measure achieved by DAGA against the single ID replay attack

Figure 4 shows that DAGA achieves the highest performance against the `not` message. However, using detection models created with higher values of n improves the detection performance against all the attack scenarios. These results also show that for the injection of non-periodic messages the detection performance reaches high values of $\mathcal{F}-$scores ($\geq$ 0.9) starting with $n = 6$, while in the other attack scenarios the detection performance are lower. In particular, the maximum $\mathcal{F}-$measure evaluated using $n-$grams of size $n = 10$ against the injection of the *low, mid,* and *top* message are $\mathcal{F} = 0.9669$, $\mathcal{F} = 0.9511$, and $\mathcal{F} = 0.9036$, respectively. These results are a clear improvement with respect to the state-of-the-art. Results presented in [56] are the same of a DAGA instance trained with $n = 2$. Hence, with its ability to increase the value of n, DAGA is able to achieve overall better detection results in this attack scenarios. The results achieved by DAGA also outperforms the detection algorithm based the CAN bus entropy [34], [57], because these approaches are only effective against high volume attacks involving the injection of hundred of messages per second.

B. Ordered sequence replay

The detection performance of DAGA against the replay of an ordered sequence of legit messages are presented in Figure 5. Figure 5 shows the $\mathcal{F}-$measure evaluated in the different tests by comparing the length of the attack (y-axis) with the value of the parameter n used for the model creation (x-axis). The cells represent the median value of the $\mathcal{F}-$measures achieved over different tests. The colors of the cells are used to highlight the range of the $\mathcal{F}-$measure, where shades of red represent poor detection results ($\mathcal{F}-$measure ≤ 0.5), shades of cyan are used to represent acceptable results ($0.5 < \mathcal{F}-$measures ≤ 0.8), and shades of green represent high detection results ($\mathcal{F}-$measure > 0.8). The right column of Figure 5 summarizes the ranges of $\mathcal{F}-$measure presented in the left matrix.

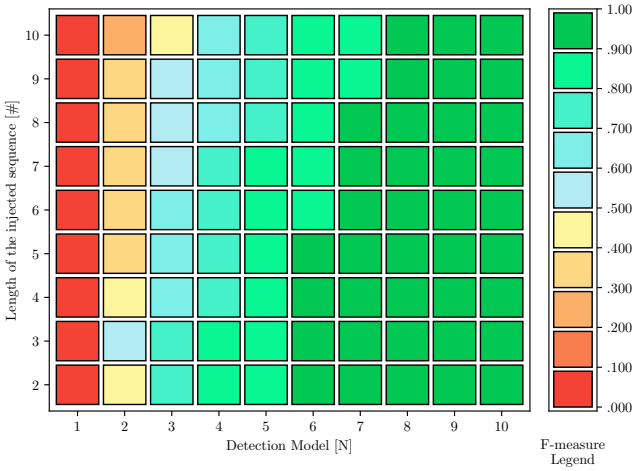

Fig. 5: Median $\mathcal{F}-$measure achieved by DAGA against the ordered sequence replay attacks

From the analysis of the detection results presented in Figure 5 it is possible to notice that by either increasing the value of the parameter n or by decreasing the length of the injected sequence the detection performance of DAGA increases. Intuitively, longer injected sequences are more difficult to detect, especially for instances of DAGA that rely on smaller values of n. Since the injected sequence is composed of valid messages, it is possible to detect anomalies only in the transitions between the normal traffic and the injected sequence, thus resulting in the detection of 2 anomalies for each attack in the best case. Moreover, models created with smaller values of n have a more limited knowledge of the evolution over time of normal message sequences, thus resulting in a higher likelihood of missing the boundaries of the injected sequence. We remark that for $n = 10$ DAGA achieves median

values of $\mathcal{F}-$measure higher than 0.98. While this attack has been documented in [8], to the best of our knowledge this is the first paper that evaluates an intrusion detection system against it, so we cannot directly compare our performance against previous work. However we observe that entropy-based anomaly detectors [34], [57] are unable to detect these low-volume attacks. Finally, although detection algorithms based message timings [12], [58] seem a suitable alternative for this type of attack, they are only applicable to periodic messages. Hence, an attacker injecting an ordered sequence of non-periodic messages could easily evade detection. On the other hand, DAGA does not rely on message timings and cannot be evaded by injecting non-periodic messages (see Section VI-A).

C. Arbitrary sequence replay

The detection performance of DAGA against the replay of an arbitrary sequence of legit messages are presented in Figure 6. The results are shown using the same structure already described for the previous attack case, depicting the median $\mathcal{F}-$measure evaluated over different tests and comparing the results according to the length of the injected sequence (y-axis) and the value of the parameter n (x-axis).

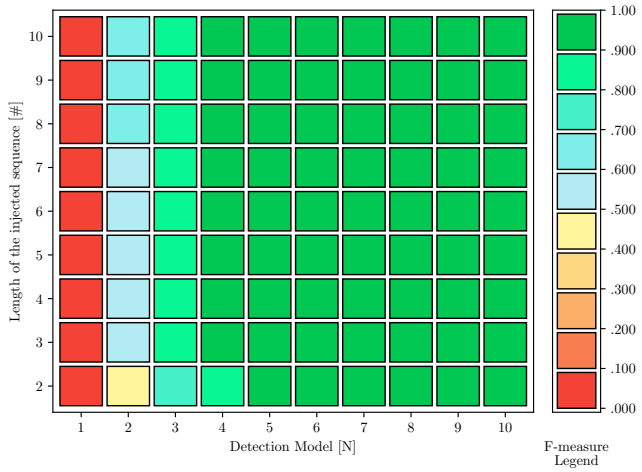

Fig. 6: Median $\mathcal{F}-$measure achieved by DAGA against the arbitrary sequence replay attacks

In case of the injection of arbitrary sequences the detection performance are mostly dependent from the value of the parameter n, where higher detection performance are achieved with higher values of n. We also remark that the length of the injected sequence does not impact the overall detection performance. The experimental evaluation of DAGA against this attack scenario shows that models created with smaller values of n are able detect the injected sequence more consistently compared to the previous scenario despite their limited knowledge of the evolution over time of normal CAN communication.

We remark that for $n \geq 5$ DAGA achieves median values of $\mathcal{F}-$measure higher than 0.95, while for $n = 10$ it achieves a median value of $\mathcal{F}-$measure of 0.99. While this attack has been also documented in [8], to the best of our knowledge this is the first paper that evaluates an intrusion detection system against it, so we cannot directly compare our performance with previous work. We remark however that the limitations of the previous proposals highlighted for the *ordered sequence replay* attack also apply to this attack scenario.

D. ID Fuzzing

The detection performance of DAGA against the injection of an invalid ID are always equal to $\mathcal{F} = 1.0$ regardless of the value of n. In case the detection model is missing a valid ID, the detection performance of DAGA will inevitably skew to lower $\mathcal{F}-$measure values. However, we remark that for the particular case of $n = 1$ it is possible to train the detection model using the formal vehicle's specifications included in DBC files. By having access to the full specifications of the vehicle (which are often shared by car makers with suppliers under *non disclosure agreements*) it is possible to create a detection model that is able to achieve 0 false positives against this attack scenario. In case the vehicle specifications are not accessible, it is possible to deploy different methodologies (such as the one presented in [59]) to map the message IDs of a target vehicle by exploiting CAN remote frames.

E. Lowest ID Denial-of-Service

The detection performance of DAGA against the lowest ID DoS attack are presented in Table II. The rows of the table represent the value of the parameter n, while the columns represent the different attack simulations (see Section IV).

n	1	2	3	4	5	6	7
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.9971	0.9971	0.9970	0.9966	0.9964	0.9970	0.9970
3	0.9991	0.9990	0.9991	0.9987	0.9990	0.9990	0.9991
4	0.9994	0.9993	0.9994	0.9993	0.9994	0.9993	0.9994
5	0.9996	0.9994	0.9996	0.9996	0.9998	0.9995	0.9995
6	1.0	1.0	1.0	1.0	1.0	1.0	1.0
7	1.0	1.0	1.0	1.0	1.0	1.0	1.0
8	1.0	1.0	1.0	1.0	1.0	1.0	1.0
9	1.0	1.0	1.0	1.0	1.0	1.0	1.0
10	1.0	1.0	1.0	1.0	1.0	1.0	1.0

TABLE II: $\mathcal{F}-$measure achieved by DAGA against the Lowest ID DoS attack

Table II shows that DAGA achieves near-optimal performance with a minimum value of $n = 2$, while optimal results ($\mathcal{F} = 1.0$) are achieved for any value of $n \geq 6$. Since this attack scenario is simulated by injecting a high volume of CAN messages, detection algorithms based on the analysis of the entropy of the CAN bus [34], [57] are also good candidates for the detection of this attack. However, the detection results presented in [57] show that to achieve a consistent detection of the DoS attack it is necessary to inject at least 100 messages per second. Since the attack frequency used for this attack scenario is of 90 messages per second (see Section IV), the algorithm presented in [57] will struggle to achieve consistent detection results, thus making DAGA the best detection algorithm against DoS. Timing-based approaches would be inapplicable for the vehicle used to collect the dataset because the message with the lowest ID is non-periodic.

VII. Implementation on reference architectures

In this section we discuss the performance of the three implementations of DAGA presented in the Section V-B. First, we describe the automotive-like boards considered in our evaluation, then we analyze the feasibility of these implementations on the boards in terms of available memory (Section VII-A) and lookup times (Section VII-B). Finally, real-time applicability for the different implementations of DAGA is discussed in Section VII-C. We remark that the three reference implementations of DAGA are prototypes used for evaluating the feasibility of DAGA on automotive microcontrollers. These tests are executed in a laboratory environment for safety purposes. Moreover, despite the DAGA prototypes are not tested on the real vehicle, we remark that by deploying any of the implementations on an automotive board equipped with any of the reference microcontrollers it is possible to achieve the same results presented in this section. To the best of our knowledge, this is the first paper in which different implementations of the same detection algorithm are tested on multiple automotive-grade boards.

We consider the five embedded boards presented in Table III: each row describes a different board, while the columns highlights the characteristic of the board that are relevant for the analysis. The first column shows the label used to address the boards, the second and third columns describe the names of the microcontrollers and the clock speed at which they operate, while the fourth and fifth columns describe the available flash and RAM memory.

The boards used in our evaluation share the same architecture of automotive microcontrollers produced by *STMicroelectronics* [60] and *NXP* [61]. The *micro #1 and #2* boards represent very simple microcontrollers (such as simple sensors and actuators) and are used as a low boundary rather than a representative hardware platform. The *micro #3* board represents common automotive microcontroller, such as the `SPC5` family produced by STMicroelectronics or the `MPC5xxx` family produced by NXP, both popular in automotive applications. The *micro #4 and #5* boards are similar to top-tier automotive boards used for infotainment or ADAS system, such as the `i.MX 8QuadPlus` family produced by NXP.

Label	Microcontroller	Clock	Flash	RAM
micro #1	ATmega328P	16MHz	32 KB	2 KB
micro #2	ATmega2560	16MHz	256 KB	8 KB
micro #3	M3 (AT91SAM3X8E)	84MHz	512 KB	96 KB
micro #4	ARMv8 - Cortex A53	1.4 GHz	16 GB*	1 GB
micro #5	ARMv8 - Cortex A72	1.5 GHz	16 GB*	4 GB

* expandable memory.

TABLE III: Technical specifications of the automotive-like boards

A. Memory requirements

The memory requirements of the detection models of the different implementations of DAGA are summarized in Table IV. The rows of the table show, for each value of the parameter n, the total number of unique $n-$grams and the memory footprint of the data structure used for the implementations of DAGA.

n	$n-$grams	Binary lookup	Hash table	sparse trie
2	1050	2.1 KB	57.49 KB	901 B
3	14945	44.84 KB	898.52 KB	13.64 KB
4	103889	415.56 KB	5.06 MB	111.88 KB
5	336048	1.69 MB	17.87 MB	479.13 KB
6	690370	4.15 MB	36.54 MB	1.28 MB
7	1128043	7.90 MB	65.42 MB	2.57 MB
8	1640341	13.13 MB	81.82 MB	4.39 MB
9	2199085	19.80 MB	129.92 MB	6.77 MB
10	2766325	27.67 MB	148.07 MB	9.68 MB

TABLE IV: Memory footprint of the implementations of DAGA

From the comparison of the memory requirements and the available *flash memory* of the microcontrollers, it is possible to determine that the *micro #4 and #5* boards support all the implementations of DAGA, while the *micro #1~#3* boards can only support a limited set of configurations. The *binary lookup* implementation is supported with n up to 2, 3, and 4 by the *micro #1, #2*, and *micro #3* boards, respectively. The *sparse trie* implementation is supported with n up to 3, 4 and 5 by the *micro #1, #2*, and *micro #3* boards. The *hash table* implementation is supported with $n = 2$ by the *micro #2 and #3* boards, but it cannot be executed on the *micro #1* board due to memory restrictions. Moreover, since the hash table implementation of DAGA is supported only with a single configuration on the *micro #1~#3* boards, the comparison of the lookup times for this implementation is limited to the only *micro #4 and #5* boards.

B. Lookup time

The lookup times of each implementation of DAGA on the *micro #1~#3* and *micro #4 and #5* boards are shown in Table V and VI, respectively. The lookup times depicted in both tables show the average and standard deviation achieved repeating the lookup operation of a valid $n-$gram $1.000.000$ times. From the comparison of the lookup times evaluated on the *micro #1~#3* boards it is clear that fastest timings are achieved using the binary lookup implementation deployed on the *micro #3* board. Despite the sparse trie implementation achieves higher lookup times, it is necessary to highlight that it is possible to deploy detection models created with higher values of the parameter n, hence achieving better detection results. By comparing the lookup times on the *micro #4 and #5* boards it is possible to notice that the sparse trie implementation offers the best lookup times for lower values of the parameter n, while the hash table structure is the fastest data structure for higher values of n.

C. Real-time application of the DAGA implementations

The real-time requirements are defined by considering the minimum inter-arrival time of the CAN messages, hence allowing DAGA to analyze a single $n-$gram in this time window. The inter-arrival times between consecutive messages on our dataset are extremely sparse, ranging from a minimum of 2 microseconds to an average of approximately 600 microseconds. Hence, we define two operational scenarios to take advantage of this variance.

Binary lookup						
	micro #1 [μs]		micro #2 [μs]		micro #3 [μs]	
n	*avg*	*dev*	*avg*	*dev*	*avg*	*dev*
2	54.08	0.0740	54.65	0.0762	8.68	0.0100
3	n/a	n/a	95.65	0.2436	16.05	0.0370
4	n/a	n/a	n/a	n/a	20.06	0.0413
sparse trie						
	micro #1 [μs]		micro #2 [μs]		micro #3 [μs]	
n	*avg*	*dev*	*avg*	*dev*	*avg*	*dev*
2	167.04	39.6438	166.98	39.6435	21.78	7.8644
3	501.27	109.2247	501.84	109.2257	50.70	14.3786
4	n/a	n/a	1121.12	248.3368	133.40	42.7876
5	n/a	n/a	n/a	n/a	315.36	84.1533

TABLE V: Lookup times of the implementations deployed on the *micro #1~#3* boards

		micro #4 [μs]		micro #5 [μs]	
	n	*avg*	*dev*	*avg*	*dev*
Binary lookup	**2**	4.12	0.0813	2.01	0.0300
	3	5.57	0.1228	2.63	0.0513
	4	6.53	0.1623	3.05	0.0870
	5	7.14	0.1840	3.38	0.0967
	6	7.52	0.1663	3.54	0.0862
	7	7.85	0.1483	3.67	0.0734
	8	7.92	0.1566	3.73	0.0846
	9	8.07	0.1143	3.80	0.0733
	10	8.20	0.1943	3.85	0.07867
Hash table	**2**	0.89	0.0809	0.62	0.0565
	3	1.01	0.1402	0.68	0.1096
	4	1.07	0.1116	0.72	0.0795
	5	1.09	0.0898	0.74	0.0877
	6	1.17	0.0881	0.78	0.0799
	7	1.26	0.1029	0.77	0.0892
	8	1.34	0.1010	0.82	0.0789
	9	1.36	0.0466	0.84	0.0708
	10	1.45	0.6760	0.85	0.0593
sparse trie	**2**	0.05	0.7634	0.02	0.5264
	3	0.26	0.9419	0.03	0.5434
	4	1.54	1.2163	0.74	0.6097
	5	3.59	1.8362	1.69	0.8876
	6	6.59	2.9241	3.24	1.1953
	7	10.53	4.3617	5.19	1.8016
	8	15.32	5.7852	7.63	2.5261
	9	20.59	7.2645	10.28	3.2965
	10	25.99	8.9467	13.15	4.2205

TABLE VI: Lookup times of the implementations deployed on the *micro #4 and #5* boards

In the first operational scenario, we require the micro-controllers to evaluate the $n-$grams within the minimum inter-arrival time of the messages (i.e. with strict real-time requirements). From the results of the lookup times evaluated on the different boards presented in Table V and VI, the only implementations that satisfy this requirement are the *hash table* implementation on both *micro #4 and #5* boards, and the *sparse trie* implementation deployed on the the same boards with a value of $n \leq 4$ and $n \leq 5$, respectively.

In the second operational scenario, we consider a relaxed bulk approach where each platform must test all messages within a particular time window. This approach allows to leverage the average timings between consecutive messages in small time frames, although a buffer is required to store the unprocessed messages in the reference time window. To this aim, we analyzed the minimum time windows required for the applicability of the boards in the relaxed bulk approach. The results of this analysis are presented in Table VII in which, for each implementation of DAGA on the supported boards, the required time window and the required buffer size are shown. These results show that in the worst scenario (i.e. the highest lookup time experimentally evaluated on the boards) the required time windows for the binary lookup implementation of DAGA range from 50 microseconds on the *micro #5* board (with $n = 10$, buffer size of 12 messages) to 3 milliseconds on the *micro #2* board ($n = 3$, buffer size of 28 messages), while with the sparse trie implementation the time windows range from 300 microseconds on the *micro #4* board ($n = 10$, buffer size of 17 messages) to 50 milliseconds on both *micro #1 and #2* boards ($n = 3$ and $n = 4$ respectively, buffer size of 98 messages).

Binary Lookup				
	time window	*messages*	*lookup [μs]*	*n*
micro #1	$2ms$	22	54.08	2
micro #2	$3ms$	28	95.65	3
micro #3	$400\mu s$	16	20.06	4
micro #4	$200\mu s$	16	8.20	10
micro #5	$50\mu s$	12	3.85	10
sparse trie				
	time window	*messages*	*lookup [μs]*	*n*
micro #1	$50ms$	98	501.27	3
micro #2	$50ms$	98	1121.12	4
micro #3	$20ms$	59	315.36	5
micro #4	$500\mu s$	18	25.99	10
micro #5	$300\mu s$	17	13.15	10

TABLE VII: Highest time windows and buffer sizes for the relaxed bulk application of the DAGA implementations

We remark that in the first operational scenario it is possible to detect anomalies in the CAN traffic after each message is sent, while the second operational scenario allows the detection of anomalies within a predictable delay equal to the used time window.

As a final consideration, despite the applicability of the proposed implementations of DAGA on micros #1~#3 seems limited, by comparing the supported implementations with their detection performance against the *single ID replay* attack (presented in Section VI-A) it is possible to achieve $\mathcal{F}-$measures equal to $0.75, 0.8$, and 0.85 on micro #1, #2, and #3 respectively, which are comparable or higher than the current state-of-the-art [34], [56], [58]. Against the *ordered sequence replay* attack (presented in Section VI-B) it is possible to achieve $\mathcal{F}-$measures in the $0.7 - 0.8, 0.8 - 0.9$, and $0.8-0.9$ range on micro #1, #2, and #3 respectively, while against the the *arbitrary sequence replay* attack (presented in Section VI-C) the results are in the $0.8 - 0.9, 0.9 - 1.0$, and $0.9-1.0$ range on micro #1, #2, and #3 respectively. Moreover, against the *lowest ID DoS* attack scenario is it possible to achieve $\mathcal{F}-$measures higher than 0.99 with all the three micros, reaching almost perfect $\mathcal{F}-$measures with micro #3 and $n = 6$ ($\mathcal{F} = 0.9998$). To the best of our knowledge, DAGA is the first detection algorithm for CAN communications tested on multiple and different platforms, deployable on boards with extremely constrained resources (such as micros #1~#3), and

able to achieve detection performance that are comparable with the current state-of-the-art against all the considered attack scenarios.

VIII. Conclusion

This paper presents *DAGA*, an anomaly detection algorithm based on the analysis of $n-$grams of CAN message IDs transmitted over in-vehicle networks. Differently from previous proposals, DAGA is designed for being applicable to resource-constrained microcontrollers, allowing the definition of multiple detection models characterized by different memory footprints. We present three reference implementations of DAGA that allow us to explore the trade-off between detection performance, detection time and hardware requirements, such as computational power and available memory. We experimentally evaluate these implementations by deploying DAGA on five different automotive-like boards, representing microcontrollers used in automotive application and characterized by similar computational power and available memory. Our evaluation considers five attack scenarios described in the related literature. Results show that DAGA achieves detection performance that are better than the related work when deployed on the more powerful hardware platforms. Moreover, DAGA can be executed on microcontrollers that are not be able to load and execute the vast majority of in-vehicle IDS approaches already presented in literature. As a final contribution, we publicly release both the source code of the three DAGA implementations and the full dataset used for the evaluation of DAGA. This dataset is composed of more than 50 hours of CAN traffic, including several attack families and variants. This dataset enables other researchers to replicate our results and foster future research in the automotive cybersecurity domain.

References

[1] P. Kleberger, T. Olovsson, and E. Jonsson, "Security aspects of the in-vehicle network in the connected car," in *IEEE Proc. Intelligent Vehicles Symp.*, 2011.

[2] C. Miller and C. Valasek, "Remote Exploitation of an Unaltered Passenger Vehicle," http://illmatics.com/RemoteCarHacking.pdf, 2015.

[3] Keen Security Lab. of Tencent, "New car hacking research: 2017, remote attack tesla motors again," Tech. Rep., 2017.

[4] M. Luo, A. C. Myers, and G. E. Suh, "Stealthy tracking of autonomous vehicles with cache side channels," in *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, Aug. 2020, pp. 859–876. [Online]. Available: https://www.usenix.org/conference/usenixsecurity20/presentation/luo

[5] S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, K. Koscher, A. Czeskis, F. Roesner, and T. Kohno, "Comprehensive experimental analyses of automotive attack surfaces," in *Proceedings of the 20th USENIX Conference on Security*, ser. SEC'11. USA: USENIX Association, 2011, p. 6.

[6] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Computing Surveys*, vol. 41, no. 3, 2009.

[7] ——, "Anomaly detection for discrete sequences: A survey," *IEEE Trans. on Knowledge and Data Engineering*, vol. 24, no. 5, 2012.

[8] C. Miller and C. Valasek, "CAN Message Injection – OG Dynamite Edition," http://illmatics.com/can%20message%20injection.pdf, 2016.

[9] R. B. GmbH, "CAN specification version 2.0," Tech. Rep., 1991.

[10] M. J. Kang and J. W. Kang, "A novel intrusion detection method using deep neural network for in-vehicle network security," in *IEEE Vehicular Tech. Conf.*, May 2016.

[11] K. Cho and K. G. Shin, "Fingerprinting electronic control units for vehicle intrusion detection," in *USENIX Security Symposium*, 2016.

[12] D. Stabili and M. Marchetti, "Detection of missing CAN messages through inter-arrival time analysis," in *2019 IEEE 90th Vehicular Technology Conf.*, Sep. 2019.

[13] M. Hosseinzadeh, B. Sinopoli, and E. Garone, "Feasibility and detection of replay attack in networked constrained cyber-physical systems," in *2019 57th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, 2019, pp. 712–717.

[14] N. Forti, G. Battistelli, L. Chisci, and B. Sinopoli, "Joint attack detection and secure state estimation of cyber-physical systems," 2019.

[15] W. Aoudi and M. Almgren, "A framework for determining robust context-aware attack-detection thresholds for cyber-physical systems," in *2021 Australasian Computer Science Week Multiconference*, ser. ACSW '21. New York, NY, USA: Association for Computing Machinery, 2021. [Online]. Available: https://doi.org/10.1145/3437378.3437393

[16] D. Stabili, R. Romagnoli, M. Marchetti, B. Sinopoli, and M. Colajanni, "Exploring the consequences of cyber attacks on powertrain cyber physical systems," 2022. [Online]. Available: https://arxiv.org/abs/2202.00743

[17] J. Kamel, M. R. Ansari, J. Petit, A. Kaiser, I. Ben Jemaa, and P. Urien, "Simulation framework for misbehavior detection in vehicular networks," *IEEE Transactions on Vehicular Technology*, 2020.

[18] F. Pollicino, D. Stabili, L. Ferretti, and M. Marchetti, "An experimental analysis of ECQV implicit certificates performance in VANETs," in *2020 IEEE 92nd Vehicular Technology Conference (VTC2020-Fall)*, 2020, pp. 1–6.

[19] F. Pollicino, D. Stabili, G. Bella, and M. Marchetti, "SixPack: Abusing ABS to avoid misbehavior detection in VANETs," in *2021 IEEE 93rd Vehicular Technology Conference (VTC2021-Spring)*, 2021, pp. 1–6.

[20] G. Dupont, J. den Hartog, S. Etalle, and A. Lekidis, "A survey of network intrusion detection systems for controller area network," in *2019 IEEE Int'l Conf. on Vehicular Electronics and Safety*, Sep. 2019.

[21] L. Ben Othmane, L. Dhulipala, M. Abdelkhalek, N. Multari, and M. Govindarasu, "On the performance of detecting injection of fabricated messages into the CAN bus," *IEEE Transactions on Dependable and Secure Computing*, pp. 1–1, 2020.

[22] M. Hanselmann, T. Strauss, K. Dormann, and H. Ulmer, "CANet: An unsupervised intrusion detection system for high dimensional CAN bus data," *IEEE Access*, vol. 8, p. 58194–58205, 2020. [Online]. Available: http://dx.doi.org/10.1109/ACCESS.2020.2982544

[23] AUTOSAR. Binary Search - Dictionary of Algorithms and Data Structures. [Online]. Available: https://www.autosar.org/

[24] ——. (2019) Requirements on Secure Onboard Communication. [Online]. Available: https://www.autosar.org/fileadmin/user_upload/standards/classic/19-11/AUTOSAR_SRS_SecureOnboardCommunication.pdf

[25] ——. (2019) Specification of Secure Onboard Communication AUTOSAR CP R19-11. [Online]. Available: https://www.autosar.org/fileadmin/user_upload/standards/classic/19-11/AUTOSAR_SWS_SecureOnboardCommunication.pdf

[26] P. Biondi, G. Bella, G. Costantino, and I. Matteucci, "Implementing can bus security by TOUCAN," in *Proceedings of the Twentieth ACM International Symposium on Mobile Ad Hoc Networking and Computing*, ser. Mobihoc '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 399–400. [Online]. Available: https://doi.org/10.1145/3323679.3326614

[27] G. Bella, P. Biondi, G. Costantino, and I. Matteucci, "CINNAMON: A module for autosar secure onboard communication," in *2020 16th European Dependable Computing Conference (EDCC)*, 2020, pp. 103–110.

[28] K. T. Cho and K. G. Shin, "Viden: Attacker identification on in-vehicle networks," arXiv 1109.1123, 2017.

[29] M. Kneib and C. Huth, "Scission: Signal characteristic-based sender identification and intrusion detection in automotive networks," in *Proc. 2018 ACM SIGSAC Conf. on Computer and Communications Security*. ACM, 2018.

[30] M. Gmiden, M. H. Gmiden, and H. Trabelsi, "An intrusion detection method for securing in-vehicle CAN bus," in *Int'l Conf. Sciences and Techniques of Automatic Control and Computer Engineering*, 2016.

[31] 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," in *CISRC '17 Proc. 12th Annual Conf. on Cyber and Information Security Research*, 2017.

[32] M. Marchetti and D. Stabili, "READ: Reverse engineering of automotive data frames," *IEEE Transactions on Information Forensics and Security*, pp. 1–1, 2018.

[33] M. D. Pesé, T. Stacer, C. A. Campos, E. Newberry, D. Chen, and K. G. Shin, "LibreCAN: Automated CAN message translator," in *Proc. 2019 ACM SIGSAC Conf. Computer and Communications Security*, 2019.
[34] M. Müter and N. Asaj, "Entropy-based anomaly detection for in-vehicle networks," in *IEEE Proc. Intelligent Vehicles Symp.*, 2011.
[35] N. Nowdehi, W. Aoudi, M. Almgren, and T. Olovsson, "CASAD: CAN-aware stealthy-attack detection for in-vehicle networks," 2019.
[36] V. Chockalingam, I. Larson, D. Lin, and S. Nofzinger, "Detecting attacks on the CAN protocol with machine learning," *Annual EECS*, vol. 588, 2016.
[37] C. Ling, "An Algorithm for Detection of Malicious Messages on CAN Buses," in *Conf. Innovative Trends in Computer Science*, 2012.
[38] R. U. D. Refat, A. A. Elkhail, and H. Malik, "A lightweight intrusion detection system for can protocol using neighborhood similarity," in *2022 7th International Conference on Data Science and Machine Learning Applications (CDMA)*, 2022, pp. 121–126.
[39] C. Y. Suen, "N-gram statistics for natural language understanding and text processing," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, no. 2, 1979.
[40] W. B. Cavnar and J. M. Trenkle, "N-gram-based text categorization," in *Proc. of 3rd Annual Symp. on Document Analysis and Information Retrieval*, 1994.
[41] C. Wressnegger, G. Schwenk, D. Arp, and K. Rieck, "A Close Look on N-grams in Intrusion Detection: Anomaly Detection vs. Classification," in *Proc. of the ACM Work. Artificial Intelligence and Security*, 2013.
[42] P. Laskov and N. Šrndić, "Static detection of malicious javascript-bearing pdf documents," in *Proc. of 27th Annual Computer Security Applications Conference*, 2011.
[43] H. Lee, K. Choi, K. Chung, J. Kim, and K. Yim, "Fuzzing CAN packets into automobiles," in *2015 IEEE 29th Int'l Conf. on Advanced Information Networking and Applications*, March 2015.
[44] K. Cho and K. G. Shin, "Error handling of in-vehicle networks makes them vulnerable," in *Proc. ACM SIGSAC Conf. Computer and Communications Security*, 2016.
[45] A. Palanca, E. Evenchick, F. Maggi, and S. Zanero, "A stealth, selective, link-layer denial-of-service attack against automotive networks," in *Proc. of 14th Int'l Conf. Detection of Intrusions and Malware and Vulnerability Assessment*, 2017.
[46] Keen Security Lab. of Tencent, "Car hacking research: Remote attack tesla motors," Tech. Rep., 2016.
[47] M. Sutton, A. Greene, and P. Amini, *Fuzzing: Brute Force Vulnerability Discovery*. Addison-Wesley Professional, 2007.
[48] CISA - Cybersecurity and Infrastructure Security Agency, "Understanding denial-of-service attacks," https://us-cert.cisa.gov/ncas/tips/ST04-015, 2019.
[49] P.-S. Murvay and B. Groza, "Dos attacks on controller area networks by fault injections from the software layer," in *Proceedings of the 12th International Conference on Availability, Reliability and Security*, ser. ARES '17. New York, NY, USA: Association for Computing Machinery, 2017. [Online]. Available: https://doi.org/10.1145/3098954.3103174
[50] T. Dagan, Y. Montvelisky, M. Marchetti, D. Stabili, M. Colajanni, and A. Wool, "Vehicle safe-mode, concept to practice limp-mode in the service of cybersecurity," *SAE International Journal of Transportation Cybersecurity and Privacy*, vol. 3, no. 1, pp. 19–39, feb 2020. [Online]. Available: https://doi.org/10.4271/11-02-02-0006
[51] PEAK System, "Pcan-usb," https://www.peak-system.com/PCAN-USB.199.0.html, Tech. Rep., 2015.
[52] Paul E. Black. Binary Search - Dictionary of Algorithms and Data Structures. [Online]. Available: https://www.nist.gov/dads/HTML/binarySearch.html
[53] B. Stroustrup, *The C++ Programming Language*, 4th ed. Addison-Wesley Professional, 2013.
[54] J. Wang, C. Lin, Y. Papakonstantinou, and S. Swanson, "An experimental study of bitmap compression vs. inverted list compression," in *Proceedings of the 2017 ACM International Conference on Management of Data*, ser. SIGMOD '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 993–1008. [Online]. Available: https://doi.org/10.1145/3035918.3064007
[55] P. van Emde Boas, R. Kaas, and E. Zijlstra, "Design and implementation of an efficient priority queue," *Mathematical systems theory*, vol. 10, no. 1, pp. 99–127, 1976.
[56] M. Marchetti and D. Stabili, "Anomaly detection of CAN bus messages through analysis of ID sequences," in *IEEE Proc. Intelligent Vehicles Symp.*, June 2017.
[57] M. Marchetti, D. Stabili, A. Guido, and M. Colajanni, "Evaluation of anomaly detection for in-vehicle networks through information-theoretic algorithms," in *IEEE 2nd Int'l Forum Research and Technologies for Society and Industry Leveraging a better tomorrow*, Sept 2016.
[58] A. Taylor, N. Japkowicz, and S. Leblanc, "Frequency-based anomaly detection for the automotive CAN bus," in *World Cong. on Industrial Control Systems Security*, Dec 2015.
[59] S. Kulandaivel, T. Goyal, A. K. Agrawal, and V. Sekar, "CANvas: Fast and inexpensive automotive network mapping," in *28th USENIX Security Symposium (USENIX Security 19)*. Santa Clara, CA: USENIX Association, Aug. 2019, pp. 389–405. [Online]. Available: https://www.usenix.org/conference/usenixsecurity19/presentation/kulandaivel
[60] STMicroelectronis. website. [Online]. Available: https://www.st.com/content/st_com/en.html
[61] NXP. website. [Online]. Available: https://www.nxp.com/

Dario Stabili is a post doctoral researcher at the Department of Engineering "Enzo Ferrari" of the University of Modena and Reggio Emilia (Italy). He received a Ph.D. in Information and Communication Technologies in 2020. His research interests include all aspects of system and network security, security for cyber-physical systems and automotive, security for automotive networks and constraint devices. Homepage: http://weblab.ing.unimo.it/people/stabili/

Luca Ferretti is a researcher at the Department of Physics, Computer Science and Mathematics of the University of Modena and Reggio Emilia (Italy). He received a Ph.D. in Information and Communication Technologies in 2016. His research focuses on applied cryptography, information security, and secure network protocols. Homepage: http://weblab.ing.unimo.it/people/ferretti/

Mauro Andreolini is an assistant professor at the Department of Physics, Computer Science and Mathematics of the University of Modena and Reggio Emilia (Italy). He received his Master Degree (summa cum laude) at the University of Roma, Tor Vergata in 2001 and his Ph.D. in 2005 from the same institution. His research focuses on design, evaluation and security of distributed and cloud-based systems, based on a best-effort service or on guaranteed levels of performance. Homepage: http://weblab.ing.unimo.it/people/andreolini

Mirco Marchetti is an associate professor at the Department of Engineering "Enzo Ferrari" of the University of Modena and Reggio Emilia (Italy), where he teaches the Masters' degree courses Cyber-Security and Automotive Cyber-Security. He received a Ph.D. in Information and Communication Technologies in 2009. His research interests include all aspects of system and network security, security for cyber-physical systems and automotive, cryptography applied to cloud security and outsourced data and services. Homepage: https://weblab.ing.unimore.it/people/marchetti