Intrusion Detection Systems on CAN Buses for Commercial Vehicles with SAE J1939 Compliant Communication

A Thesis Submitted for obtaining
the Scientific Title of PhD in Engineering
from
Politehnica University Timișoara
in the Field of
Computer and Information Technology
by

Eng. Camil-Vasile JICHICI

PhD Committee Chair:	Prof. Dr. Eng. Mihai Victor MICEA
PhD Supervisor:	Prof. Dr. Eng. Bogdan-Ioan GROZA
Scientific Reviewers:	Prof. Dr. Eng. Ion BICA
	Prof. Dr. Eng. Gheorghe SEBESTYEN-PAL
	Prof. Dr. Eng. Gheorghe-Daniel ANDREESCU

Date of the PhD Thesis Defense: 20-October-2023

Acknowledgement

I would especially like to thank Prof. Dr. Eng. Bogdan Groza, my PhD advisor, for his support of my research endeavors during my PhD studies. I want to express my gratitude for his willingness to share his knowledge and experience on automotive security, and also for supporting and encouraging me throughout the whole process of writing this thesis. I want to thank him for the calmness, patience, and availability he has shown throughout these 5 years. Additionally, I thank him for giving me the opportunity to participate in teaching activities as an assistant at UPT and for including me in the research activities for the PRESENCE project.

A special thanks goes out to Radu Ragobete, my colleague from Continental, for sharing his insightful knowledge of microcontrollers with a particular emphasis on low-level drivers, and to Adriana Berdich, my PhD colleague at UPT, for her valuable experience on control systems.

I want to thank my PhD colleagues Lucian Popa, Tudor Andreica, and Adrian Musuroi for their excellent cooperation. I would also like to thank the members of the guidance committee: Associate Professor Pal-Stefan Murvay, Lecturers Raul Robu and Horatiu Gurban, for our productive working relationship and the support they provided.

Last but not least, I am grateful to my family for their continuous support during my PhD studies.

Timişoara, August 2023 Camil-Vasile JICHICI

Jichici, Camil-Vasile

Intrusion Detection Systems on CAN Buses for Commercial Vehicles with SAE J1939 Compliant Communication

Keywords:
CAN bus, vehicle security, intrusion detection and prevention, SAE J1939

Abstract:

The SAE J1939 protocol, built on the top of the CAN protocol, is a standard for heavy-duty in-vehicle networks. This commercial vehicle sector plays a significant role in various domains, including goods distribution, public transportation, construction, agriculture, forestry and marine vehicular technologies, etc., all of these being essential for the global economy. Given the high degree of inter-connectivity of modern vehicles and the numerous cyber-attacks reported in the past decade, detecting and preventing intrusions on J1939 communications is crucial. In the light of the above, this thesis proposes various intrusion detection systems for CAN buses focusing on the SAE J1939 heavy-duty vehicle buses. The techniques behind the design and implementation of these IDS varies from the use of machine learning algorithms, to deterring adversaries by concealing the content of CAN frames using symmetric encryption, or performing a fine-grained analysis at the control system level. A novel mechanism to decode the content of the CAN frames, ID and data field is introduced, which paves the way for real-time destruction of the intrusions before the complete reception of malicious frames. Also, a more in-depth analysis performed at control system level opens the road for complementing the traditional CAN bus attacks with more knowledgeable attacks that can evade the intrusion detection and for designing mitigation mechanisms to detect such attacks. The experimental part builds on realistic frameworks deployed within an industry-standard tool, i.e., the CANoe environment, which allows for the integration of adversary models and intrusion detection.

Contents

Chapter 1

Introduction

1.1 Motivation

The demand for human commuting and the delivery of goods is growing at an increasing rate. This brings a significant increase in the number of vehicles and an expansion of the transportation infrastructure. It is no surprise that between 2000 and 2021 there was an increase in the number of registered heavy-duty trucks in the United States of almost 93.5% [1]. Given the above, heavy-duty vehicles such as buses, tractors or trailers, have a significant role. Millions of heavy-duty vehicles travel hundreds of kilometers on highways every day and the likelihood of incidents increases, hence constantly boosting the safety of the traffic participants is crucial. Furthermore, the commercial vehicle sector may also have other applications such as agriculture where the automation of the agricultural machinery (tractors, combines, etc.) is essential since it can enhance the efficiency, productivity, product quality and sustainability [2]. According to [3] over 2.45 million of agricultural machinery were sold globally in 2020. This reflects an increase compared to the previous year (2019) of around 4.1%. The sale of equipment in the agricultural sector is anticipated to increase over the next years, reaching 2.7 million units in 2029. The safety mechanisms are of great importance for this sector as well.

Regarding safety, several improved driver assistance systems, including forward collision warning, lane departure warning, alcohol ignition interlock devices, autonomous emergency braking and blind-spot recognition, have been developed over the past ten years. The ability of vehicles to drive autonomously, without human involvement, is a short-term goal that is especially important in the context of heavy-duty vehicles because of the significantly larger distances they must travel. To fulfill these features, heavy-duty vehicles are also turning into intricate cyber-physical systems with millions of lines of code running through dozens of Electronic Control Units (ECUs) and a variety of sensors, actuators, cameras, and radars. In-vehicle ECUs currently communicate with one another over a variety of communication interfaces, including the traditional Controller Area

Network (CAN), FlexRay, and the more recent 100BaseT1 Ethernet, which supports substantially higher communication speeds. However, in spite of recent advancements, CAN is still by far the most widely employed communication medium since it offers a very good cost-performance ratio and a reliable solution for the majority of real-time applications. Two updated versions of it, i.e., CAN-FD [4] as well as CAN-XL [5] enable much higher bit rates and larger payloads, which ensure the longevity of the CAN protocol for the future decades.

On the other hand, due to the increase of software-driven features, connectivity and semi-automated functions, vehicles become vulnerable to cybersecurity attacks. As a result, several attacks were reported by the researchers in various publications, e.g., [6], [7], and [8]. These adversarial interventions may have catastrophic effects for both vehicle occupants and traffic participants alike. Koscher et. al. [6] show that by mounting CAN bus attacks on real world vehicles it is possible to take the control of several critical automotive functions while completely ignoring the driver input. Examples of such actions include disabling the brakes, unlock the car, stopping the engine, etc. Due to the fact that the attacks demonstrated in [6] require a physical connection to the CAN bus, their impact may be more limited. But in addition to this, the same authors proved in a subsequent work [7] that similar attacks can be conducted remotely by establishing a connection with mechanics tools or infotainment units using wireless interfaces such as Bluetooth or WiFi. A comprehensive research on attack surfaces is also provided by Miller and Valasek in [8]. Foster et al. explore the use of a Telematic Control Unit (TCU) device, which is connected to the OBD port, and can be remotely compromised. This TCU enables the access to the CAN bus of the vehicle and is able to inject malicious CAN frames targeted to different ECUs and thus interactively control the vehicular systems from arbitrary distances [9]. The web browser of a TESLA car was successfully exploited in a remote attack performed by Nie et al. [10]. By leveraging the over-the-air (OTA) software update mechanism, the same authors were able to remotely compromise various safety-critical ECUs from TESLA cars [11]. A more recent work [12] provides a comprehensive analysis on the security of 77 wireless OBD dongles acquired from Amazon. The authors developed an automated tool that facilitates testing these dongles on a real-world vehicle against several attack scenarios that can be mounted by an adversary. Consequently, these dongles expose an attack surface that can be exploited by injecting malicious CAN frames into the genuine CAN traffic of a vehicle. A practical wireless attack that targets the CAN communication is proposed in [13]. The authors of [13] exploit the connection between internal networks such as CAN and external networks such as 3G or 4G mobile networks, which opens the road for an adversary to mount a long-range wireless attack targeting CAN vulnerabilities.

Regardless of the attack access point, the lack of security of the CAN bus, which was introduced by BOSCH in the 80s [14], is the root cause for the previously mentioned attacks. The SAE J1939 standard [15], developed by the Society of Automotive Engineers in the 90s, is dedicated to CAN communication within heavy-duty vehicles and complements the conventional CAN bus protocol by introducing specific features which are detailed in Chapter 3. Since the SAE J1939 CAN protocol is built on top of the standard CAN protocol, similar attacks to those previously described can also be mounted on J1939 compliant CAN buses. A first example of such attacks on J1939 CAN buses was proposed in [16]. Burakova et al. [16] demonstrate by practical experiments on two J1939 compliant vehicles (a semi-tractor as well as a school bus), that safety-critical attacks, e.g., truck acceleration while it is moving, turning off the engine brake, can be mounted through J1939 specific diagnostic port. The authors from [17] are the first that address attacks on the SAE J1939 specific transport protocols and demonstrate a DoS attack that targets multi-packet transmissions. A setup in which the adversary is connected to a J1939 compliant CAN bus and has the ability to intercept and inject malicious frames inside a heavy-duty vehicle via the J1939 specific OBD port is depicted in Figure 1.1.

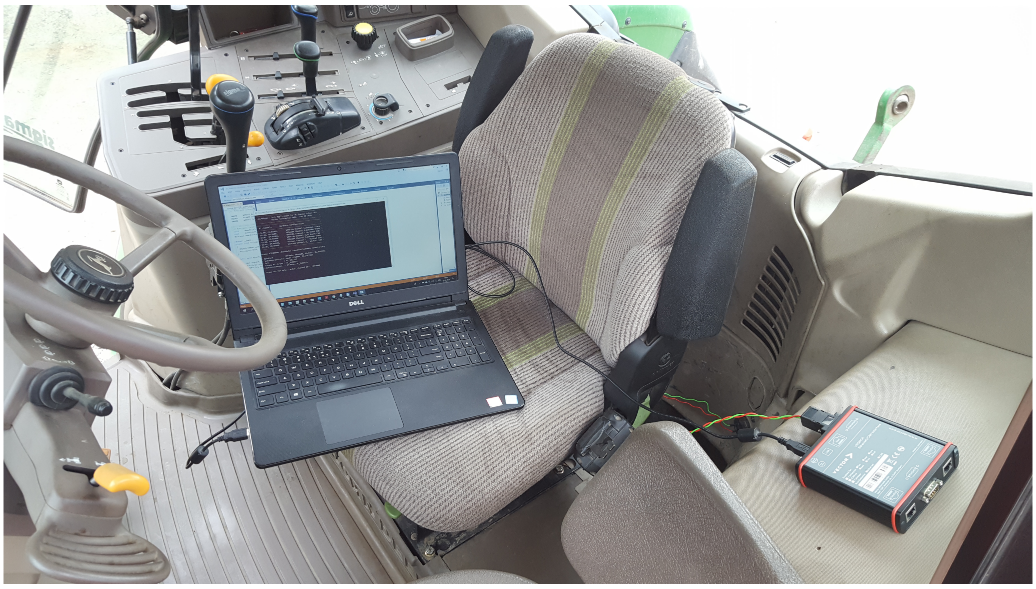


Figure 1.1: A typical heavy-duty vehicle implementing the SAE J1939 standard and tools for data collection of the SAE J1939 CAN traffic

Considering the concerns for the protection of numerous passengers (inside buses), or goods (inside heavy-duty trucks), the security solutions employed in the commercial vehicle sector require a special attention. The motivation behind this thesis comes in the light of the above. In this respect, this thesis pursues the deployment of intrusion detection systems tailored to J1939 CAN buses.

1.2 Research objectives

This section gives an overview of the research objectives from this thesis. The primary research objective is the deployment of intrusion detection systems (IDSs) customized to SAE J1939 protocol, which is built on top of the CAN protocol and defines the upper layers from the communication stack, e.g., data link layer, network management layer, etc., for implementing in-vehicle functionalities over CAN buses. But before moving to J1939 specific solutions, some investigations on intrusion detection mechanism and their integration for regular CAN bus traffic is also necessary. More precisely, the major research objectives of this thesis, can be summed up as follows:

1. Review of the literature on relevant works for intrusion detection on CAN buses;
2. CAN traffic extraction from real-world passenger cars as well as J1939 compliant vehicles for experimental purposes;
3. Performance assessment of machine learning algorithms as candidates for IDS on CAN buses as well as the feasibility of using such detection mechanisms in real-life vehicular applications;
4. Design and implementation of a framework which allows for the simulation of adversarial models and intrusion detection in CANoe;
5. Design and implementation of an intrusion detection and prevention system targeting J1939 CAN buses;
6. Analysis of attacks performed at the control system level on J1939 compliant CAN buses and the design of the appropriate countermeasures.

1.3 Major contributions

An overview of the contributions of this thesis is provided in this section. This thesis discusses several IDS approaches for CAN buses, with a focus on J1939 compliant CAN buses in particular. In light of the stated research objectives, the contributions of the author can be summed up as follows:

1. CAN traffic was collected by the author from several vehicles, specifically, 427,660 CAN frames collected from a heavy duty-vehicle in motion that is compliant to SAE J1939 communication [18] and was directly used for the experiments in this thesis;
2. CAN traffic was also collected by the author for several works which he co-authored or partly used in his papers as a first author, specifically, 2,488,248 CAN frames

collected from three different Dacia Dusters [19], [20]; 2,783,265 CAN frames and 90,723 voltage bit samples collected from 5 passenger cars [21]; 154,779 CAN frames and 4,021 voltage bit samples collected from a heavy duty vehicle compliant to SAE J1939 communication [21];

3. Evaluation of neural networks in detecting intrusions on CAN as well as their computational performance on automotive embedded platforms [22];
4. Implementation and testing of a framework that allows the integration of adversary models and intrusion detection systems in an industry standard environment, i.e., CANoe [19];
5. Deployment of an intrusion detection and prevention system tailored to meet J1939 specifications. To demonstrate the correctness as well as the feasibility of using the suggested solution in real-world vehicles, a proof-of concept implementation in a laboratory setup is provided [18];
6. A special implementation for decoding the content of CAN frames before the receivers have set the acknowledgment bit. This makes it possible to instantly discard the intrusions with no need for specialized hardware [18];
7. A proposal for a detection mechanism on attacks performed at the control system level that are challenging to be detected by a traditional IDS [23];
8. Development of an experimental setup that connects the two most widely used tools in industry, CANoe for the simulation of in-vehicle networks and MATLAB for control system design, respectively [23].

These contributions are outlined by a number of scientific articles published in different journals and conference proceedings as well. The application of neural networks as possible candidates for IDSs in CAN was explored by the author in [22]. The majority of the experiments were carried on traces from a J1939 simulation and a small part of them were performed on other public datasets. The runtime of the detection algorithm was evaluated on three automotive embedded platforms in order to determine whether the proposed solution could be used in real-world scenarios. One of the platforms represents the low-end device group, while the other two are high-end device candidates. Given the fact that the attacks on J1939 CAN traffic from [22] were generated using a C# script, a more realistic scenario would be to use real-world CAN traffic and to augment it with injections using a CANoe simulation. Because of this, the author analyzes this scenario in [19] where a framework is offered to replay the CAN traffic collected from actual vehicles and permit the integration of adversary models along with detection systems inside a CANoe-based simulation. The k-NN classifier was investigated as a candidate for IDS in this work [19], although the framework was designed to support any other IDS method. The next two papers of the author [18], [23] address IDS in the context of the SAE J1939

CAN buses. In the first one [18], the author proposes a two-stage IDS complemented by a prevention mechanism. In the first stage, the validity of the encrypted addresses is verified. The second stage performs appropriate range checks to identify single bit changes in the payload. Since the payloads of CAN messages are encrypted, the avalanche effect of block ciphers makes it easier to spot adversary interventions. The prevention mechanism requires the decoding of each CAN message (ID and payload) before the receivers confirm the correct reception by overwriting a dominant bit in the ACK slot. Then, if the current CAN frame is regarded as intrusion, it is then discarded by forcing an error frame. To demonstrate the practical applicability of the solution, a proof-of concept implementation on high-end in-vehicle controllers is provided. Last but not least, in [23], the author examines attacks on J1939 CAN buses mounted at control system level and proposes a detection mechanism for such adversarial manipulation. Additionally, this research shows how poorly the machine learning based IDSs performs in identifying these kind of subtle payload alterations and that change detection mechanisms are far more effective.

In addition to the aforementioned research papers, which represent the core of this thesis, the author has been involved in several other research papers, five of which, are also related to automotive cybersecurity and one which addresses mobile device pairing based on the environmental data. Concretely, an efficient approach for localizing the CAN network nodes based on propagation delays of physical CAN signals has been investigated in [24]. Two physical connections, one at either end of the bus, and merely one rising edge are required to examine the propagation delays. In this work, the author has contributed with voltage data collection based on a laboratory setup as well as with the preparation of the data collection setup inside a real vehicle, i.e., Renault Megane. A comparative performance between Android head units and automotive graded controllers for the deployment of several binary classifiers as candidates for IDSs is discussed in [20]. The author's contribution to this work accounts for the CAN traffic collection from a real-world SUV, i.e, Dacia Duster as well as for the augmentation of the collected traffic with specific attacks inside a CANoe based simulation. Another work which is under submission addresses a concept for a cloud-based IDS that makes use of the cloud infrastructure and the computationally powerful Android head units employed in current vehicles. The concept also includes an incident response team that performs additional evaluation of the detection results and the final outcome of this is recorded on the Blockchain as reports that comply with the ISO/SAE 21434 standard [25]. Here the CAN traffic was collected from three different Dacia Duster vehicles in order to demonstrate the transfer learning capability of the proposed IDS. Physical fingerprinting of the electronic control units (ECUs) based on clock skews and voltage data is proposed in [21]. The performed analysis led to the identification of 54 ECU fingerprints. An extensive dataset, including skew and voltage data collected from 9 passenger cars and a J1939 compliant vehicle as well, was used for the evaluation. The data collected from 5 out of 9 passenger cars and from the J1939 heavy-duty vehicle were collected by the author. The author also contributed to a paper where the influence of wiring characteristics on CAN voltage fingerprints is investigated [26]. As a result of this research, it has been determined that CAN networks that rely on commercial or industrial cables have higher noise and slew rates than those that rely on automotive graded cables. Last but not least, the author was a member of the PRESENCE project were he also contributed to a work that addresses the secure device pairing under multi-modal transport based on environmental data, i.e, accelerometer data [27].

Overall, the author has contributed to 10 scientific articles, out of which 9 are focused on automotive security and one is related to secure pairing of mobile devices based on accelerometer data:

1. Camil Jichici, Bogdan Groza, and Pal-Stefan Murvay, "Examining the Use of Neural Networks for Intrusion Detection in Controller Area Networks", Innovative Security Solutions for Information Technology and Communications: 11th International Conference, SecITC 2018, Bucharest, Romania, November 8–9, 2018, Springer International Publishing, 2019,

2. Camil Jichici, Bogdan Groza, and Pal-Stefan Murvay, "Integrating Adversary Models and Intrusion Detection Systems for In-Vehicle Networks in CANoe", Innovative Security Solutions for Information Technology and Communications: 12th International Conference, SecITC 2019, Bucharest, Romania, November 14–15, 2019, Springer International Publishing, 2020,

3. Camil Jichici, Bogdan Groza, Radu Ragobete, Pal-Stefan Murvay and Tudor Andreica, "Effective intrusion detection and prevention for the commercial vehicle SAE J1939 CAN bus", IEEE Transactions on Intelligent Transportation Systems, vol. 13, pp. 17425-17439, 2022,

4. Camil Jichici, Adriana Berdich, Adrian Musuroi, and Bogdan Groza, "Control System Level Intrusion Detection on J1939 Heavy-Duty Vehicle Buses", IEEE Transactions on Industrial Informatics, 2023,

5. Bogdan Groza, Lucian Popa, Pal-Stefan Murvay and Camil Jichici, "CAN-SQUARE-Decimeter Level Localization of Electronic Control Units on CAN Buses", Computer Security–ESORICS 2021: 26th European Symposium on Research in Computer Security, Darmstadt, Germany, October 4–8, 2021, Springer International Publishing,

6. Lucian Popa, Bogdan Groza, Camil Jichici, and Pal-Stefan Murvay, "ECUPrint - Physical Fingerprinting Electronic Control Units on CAN Buses Inside Cars and SAE J1939 Compliant Vehicles", IEEE Transactions on Information Forensics and Security, vol. 17, pp. 1185–1200, 2022,

7. Tudor Andreica, Christian-Daniel Curiac, Camil Jichici, and Bogdan Groza, "Android Head Units vs. In-Vehicle ECUs: Performance Assessment for Deploying In-Vehicle Intrusion Detection Systems for the CAN Bus," IEEE Access, vol. 10, pp. 95161–95178, 2022,

8. Lucian Popa, Camil Jichici, Tudor Andreica, Pal-Stefan Murvay and Bogdan Groza, "Impact of Wiring Characteristics on Voltage-based Fingerprinting in Controller Area Networks", IEEE 17th International Symposium on Applied Computational Intelligence and Informatics (SACI 2023), 2023,

9. Tudor Andreica, Adrian Musuroi, Alfred Anistoroaiei, Camil Jichici and Bogdan Groza, "Blockchain Integration for in-Vehicle CAN Bus Intrusion Detection Systems with ISO/SAE 21434 Compliant Reporting", **(under submission)**,

10. Bogdan Groza, Adriana Berdich, Camil Jichici, and Rene Mayrhofer, "Secure Accelerometer-Based Pairing of Mobile Devices in Multi-Modal Transport", IEEE Access, vol. 8, pp. 9246–9259, 2020.

1.4 Thesis organization

The structure of the thesis is explained in this section. A literature review on works that address automotive security focusing on CAN IDS is discussed in Chapter 2. Chapter 3 provides an overview on CAN protocol and SAE J1939 specifics and finally introduces the metrics required to evaluate the accuracy results in detecting intrusions for the proposed solutions. In Chapter 4 a neural network based IDS is proposed. This chapter presents results on detecting intrusions as well as computational results on different automotive graded controllers. Chapter 5 addresses a framework developed in CANoe which facilitates the integration of adversary models and intrusion detection as well inside a simulation. As an example, the integration of an IDS based on k-NN is shown. The IDS is then evaluated using the defined adversarial models. A two-stage IDS specifically designed to meet J1939 specification along with a prevention system that permits the instantaneous destruction of CAN messages that have been manipulated by an adversary are addressed in Chapter 6. The laboratory setup designed to deploy and test the proposed intrusion detection and prevention system using J1939 compliant CAN traffic collected from a heavy-duty truck, is also described. In Chapter 7 a control system level intrusion detection on SAE J1939 CAN buses is proposed. Chapter 8 holds the conclusion of this thesis.

Chapter 2

Literature review and metrics for intrusion detection

2.1 Related work on CAN security

This section provides a summary of the relevant works related to authentication methods and intrusion detection systems for CAN buses. These are discussed as a background for the next section, which are solely focused on J1939 CAN buses.

2.1.1 Authentication based approaches

A large number of works emerged to counter the attacks as stated in the introduction section. Most of the proposed solutions account for the authentication of CAN messages by using truncated Message Authentication Codes (MAC), e.g., [28], [29], [30]. A CAN frame and the associated authentication tag must be transmitted in order to perform message authentication. Given that a CAN frame can hold up to 8 data bytes, accommodating a MAC code inside the payload is challenging. Consequently, most of the works truncate the MAC before integrating the tags inside the payload [31], [32], [33]. Other works decide to split the bits of the authentication tag between the payload and the ID [34], [35]. This generally requires the use of extended CAN format and since, for example, 18-bits of the MAC will be packed into the extended ID, the payload is less loaded with security elements. The use of a second frame that carries the MAC tag is also discussed in [36] and [37]. However, this will result in a significant increase of the busload. The authors of [38] suggest to authenticate only the CAN frames that are related to safety-critical functions in order to minimize the busload.

2.1.2 Intrusion detection systems

In computer networks, an IDS is a component (hardware or software) that monitors the network for malicious activity. In regards to this, relevant IDSs that address CAN security were proposed and are detailed next. The authors of [39] provides an in-depth analysis of the most recent technical challenges for embedded systems inside vehicles, addressing standards, techniques, hardware and software solutions, network topologies, functional safety concepts, and security design strategies. A strong body of research provides an extensive and structured overview of vulnerabilities of in-vehicle networks, adversarial manipulations and security measures to mitigate them [40], [41], [42]. Similar efforts are made in the works [43], [44], [45], but the focus is concentrated on IDS solutions.

Considering that most of the CAN frames are periodically transmitted in accordance with the frequencies that the manufacturer has specified for them, any deviation from this cycle time can be used to detect intrusions [46], [47] [48], [49] and [50]. Obviously, if a DoS attack is mounted on the CAN bus, the periodicity of regular traffic exhibits a different behavior compared to the normal one. The major drawback of such solutions comes from CAN frames that have a spontaneous transmission, i.e., frames that are triggered on event, since in this case periodicity cannot be used to distinguish between frames that are manipulated by an adversary and the ones that are part of legitimate CAN traffic. Marchetti et al. propose an IDS that relies on a transition matrix, which is used to store patterns for legitimate CAN frame sequences and any deviation from such sequences is regarded as an intrusion [51]. Moreover, the proposed solution is suitable for deployment on resource-constraint devices, in-vehicle ECUs, due to its reduced memory and processing requirements [51]. A similar approach is evaluated in [52] to detect low-rate injection attacks in Control Area Network (CAN). The use of decision tree classifiers (DTC) is explored in [53]. The inputs for the DTC are based on the entropy computed over the number of IDs that occur in a time window.

Apart from the IDS solutions that rely on the frequency of the CAN traffic, there are also relevant proposals that examine the payload of the CAN frames. The use of Hamming distances to detect anomalies inside the payloads is evaluated in [54]. The proposed solution consists of a preliminary stage as well as a detection stage. The authors determine for each frame identifier the message validity limits – these represent the minimum and maximum Hamming distances calculated between consecutive CAN frames that share the same ID using 20% of the CAN traffic. The remaining messages are employed to test the proposed IDS. An intrusion is detected when the distance exceeds the valid ranges. Since the meaning of the CAN signals packed inside the payloads is not made public by the OEMs or defined in the standards, the authors of [55] perform an in-depth analysis of the data fields from the CAN traffic. As a result, they define four semantically-meaningful formats: constant field, multi-value field, counter field as well as sensor field, and split the content of the payloads according to these formats using a greedy algorithm. Based on this separation the authors train and build a model using Ternary Content-Addressable Memory (TCAM). In the testing phase, any anomaly from the model rules is considered as malicious [55].

Other lines of work are focused on machine learning based deployments to detect intrusions. The use of recurrent neural networks with Long Short-Term Memory (LSTM) to detect CAN bus intrusions is evaluated in [56]. The authors collected the CAN traffic for 19 hours from a 2012 Subaru Impreza in driving conditions, 13 hours of data is employed for training and the rest for testing. In the training state, each input accounts for the payload, i.e., 64 bit-values. This approach demonstrates good performance in detecting intrusions on the CAN bus. Similarly, LSTM-based IDSs were evaluated in the works [57] and [58]. The authors of [59] propose an IDS that relies on a deep neural network (DNN). The structure of the DNN has been originally proposed in [60] and employs the rectified linear unit (ReLU) as activation function. The experiments from this work [59] are carried out on CAN traffic generated using a software tool OCTANE [61]. Similar to previous work, the 64 bits of the payload serve as inputs for the DNN. These two works [58] and [59] do not take into account the frequency of the CAN frames during the training stage, hence they are unable to detect replay attacks. The use of deep convolutional neural networks (DCNN) is explored in [62], while the use of k-NN and SVM is evaluated in [63] for detecting DoS and fuzzing attacks.

Nevertheless, there are also proposals that rely on both frequency and payload of CAN frames. Groza et al. examine the use of Bloom filters to detect replay and modification attacks on the CAN bus [64]. The proposed IDS monitors the frequency of the messages and payload for each CAN ID. The use of entropy computed over CAN frames is explored in [65] and [66]. The authors of [67] discuss an IDS proposal that relies on a statistical model, i.e., Hidden Markov model and its observations. Dong et al. [67] characterize the anomaly detection problem as a multi-observation HMM. This determines the abnormal state of a CAN message by estimating the likelihood of a frame occurring at a given time with a certain ID and payload. In [68], the use of neural networks that take as inputs the recurrent plots, is discussed.

It is also worth mentioning that a strong body of research is focused on the use of physical properties for IDS deployments. Typically, in-vehicle controllers operate inside vehicles using local clocks which exhibit certain skews. In this respect, the research in [69] proposes an IDS that relies on the clock-skew fingerprints. The rationale is to estimate the clock skew for each ECU based on the cyclic CAN frames they send and further create a baseline for the clock behavior. The IDS employs the CUSUM (Cumulative Sum) in order to spot anomalous shifts from this baseline. However, further research [70], [71] proved that the clock skew of a target ECU can be reproduced by an adversary node in order to evade the detection mechanism. Moreover, such clock based fingerprinting methods can be affected by temperature. Consequently, the authors of [72] analyze the relationship between temperature and clock offset fluctuations in the context of sender identification and intrusion detection. Murvay and Groza analyze the CAN electric signal characteristics in order to uniquely identify each potential sender node [73]. The same authors use the propagation delays of physical CAN signals to localize and detect intruders [74]. A more recent concern of the research community is related to ECU fingerprinting using voltage based features. In this context, Cho et al. [75] employ the voltage profiles, which are created based on voltage measurements of CAN frames, as well as the ACK voltage thresholds in order to fingerprint ECUs and set the room for the detection of intruder ECUs. The authors test the proposed methodology using an experimental setup and two real-world vehicles. A similar work is proposed in [76], however, this requires the use of extended CAN format to circumvent the arbitration phase. The rationale behind choosing the extended format is as follows. The electric signals of the last 18 bits, in the case of 29-bit extended ID fields, are driven only by a single ECU, while in the case of a standard CAN frame, some bits of the identifier can be generated from multiple ECUs engaged in the arbitration phase. To overcome this limitation [76], a different approach for monitoring the voltage signals is discussed in [69]. This accounts for the inspection of state transition between recessive to dominant or vice-versa. The lack of consideration for environmental factors like temperature and battery voltage level may be a downside of these proposals. An edge based identification, which takes into account temperature and battery voltage variations is discussed in [77].

2.2 Related work on J1939 security

This section provides a brief overview of relevant works that address the vulnerabilities of the SAE J1939 protocol and the proposed solutions to mitigate them. The authors of [78] compare the cybersecurity risks exposed by the heavy-duty vehicles to those exposed by the regular cars in order to identify and assess potential cybersecurity threats and hazards that could influence the dependability, safety, and operability of heavy-duty vehicles. Murvay et al. investigate the shortcomings of the SAE J1939 specific features, which open the road for several attacks such as: denial of service (DoS), distributed denial of service (DdoS), and impersonation. [79]. A comprehensive analysis of the vulnerabilities exposed by the *Broadcast Data Transfer* and *Connection Mode Data Transfer*, which are employed as transport protocols in the SAE J1939 CAN networks, is presented in [80]. The experiments are conducted by the authors using a real-world heavy-duty truck, i.e., a 2014 Kenworth T270 Class 6 either in the stationary or moving scenarios [80].

In contrast to the context of passenger cars, where several efforts were done as a response to the reported CAN bus attacks, in the context of heavy-duty vehicles, only a few recent works started to address the security of J1939 CAN buses. Another difference between the two sectors, regular cars and heavy-duty vehicles, is regarding the authentication of CAN frames. For the regular cars, some of the proposed solutions rely on the use of cryptographic MACs (Message Authentication Codes) inside the payload to authenticate each CAN message, while for the heavy-duty vehicles these cannot be integrated due to the fully allocation of the payload [81]. Therefore, such an authentication mechanism of each CAN frame, can be only performed by using an extra frame [82], but this will effectively double the busload. Considering the above, the research community explored different options, which are detailed below.

The use of PKI (public key infrastructure) as an authentication mechanism for J1939 CAN traffic was suggested in [79]. Recently, the authors of [83] propose two authentication mechanisms for J1939 buses with CAN and CAN-FD communication. One of them is dedicated for resource-constraint nodes and the other for nodes with more computational resources. Both of them require an authentication key exchange protocol as well as a protocol responsible for time synchronization. The difference between the two is that for the first (resource-constraint nodes) the key-exchange is performed using one-pass authentication and for the second (powerful nodes), which support PKI, certificateless signature scheme is used [83]. A solution for protecting CAN J1939 buses that employs the secure CAN transceiver, which allows to define a whitelist as well as a blacklist of frame identifiers, is proposed in [84]. Besides employing the secure CAN transceiver, which was introduced by NXP [85], the authors of [84] also account for the use of CMACs (Cipher-based Message Authentication Codes) that are computed and sent periodically, i.e, at each second, to detect any message alterations that are transmitted during this time. In [86], the use of strong FIPS 140-2 encryption applied on the SAE J1939 diagnostic traffic exposed between the diagnostic tool and a laptop, which communicate either using wired or wireless connections, is explored. A similar work accounts for the AES-128 encryption of the traffic between the diagnostic gateway ECU, which is responsible to collect diagnostic faults from the other ECUs, and vehicle diagnostic adapter system, i.e, tester and external dedicated tools [87].

The Technology and Maintenance Council (TMC) recommends the use of RP1210 devices for re-flashing the SW on the ECUs or examining emissions related to the ECUs inside heavy-duty vehicles [88]. However, very recently, the authors of [89] demonstrated that the data passing via RP1210-based diagnostic systems is susceptible to man-in-the-middle (MitM) attacks mounted from the host diagnostics computer and propose the use of machine authentication codes or authenticated encryption using the previously exchanged keys between communication parties as a mitigation strategy for these attacks. The viability of a specific attack that targets Address Claiming Procedure is demonstrate in [90]. Basically, this attack disables the CAN communication to and from a target ECU using a single CAN frame. The authors of [90] also propose an active defense mechanism that enables both detection of such attack by using a bit-banged CAN filter and destruction of the malicious frame with an error flag once an attack is discovered. Following the introduction of the J1939 CAN-FD networks [91], the Society of Automotive Engineers also proposed the Secure On-board Communications with optional Encryption (SecOC/E) for protecting them [92]. The SecOC/E accounts for the authentication of the ECUs by using digital certificates and a secret key is exchanged between ECUs to further encrypt the relevant J1939 CAN-FD frames. A performance evaluation of a vehicle system, which implements the SecOC/E, is provided in [93].

The implementation of the IDS for protecting CAN buses is of great interest to the research community, and the J1939 CAN buses are no exception. Shirazi et al. propose a supervised machine learning based IDS that relies on various traditional classifiers including Decision Tree, Gaussian Naïve Bayes, Gradient Boosting, K-Nearest-Neighbors, Random Forrest as well as Support Vector Machine (SVM) to spot DoS and fuzzing attacks [94]. An unsupervised learning approach based on hierarchical agglomerative clustering is investigated in [95]. This IDS is designed to learn the normal operation of the vehicle based on CAN packets and use that knowledge to detect the malicious CAN traffic [95]. An anomaly detection approach for J1939 CAN buses, which is based on a precedence graph-based, is explored in [96]. Recently, the authors of [97] propose a solution that relies on the periodicity of the frames, on a dependency tree built using the correlation between the payloads, as well as on the voltage techniques to detect spoofing, masquerade, and manipulation attacks in J1939 and NMEA2000 networks. The voltage analysis permits to detect manipulation attacks, even single bit manipulation by examining unusual changes in the electric potential when a transition from a dominant to a passive state occurs [97].

Some effort was also conducted towards testing the adversarial manipulations and security mechanisms as well, using dedicated frameworks and automotive specific test strategies. A verification and validation methodology is provided by Hariharan et al. [98]. This is based on a security testing strategy using fuzzing and penetration tests specifically designed for J1939 heavy-duty in-vehicle buses. Another work accounts for a testbed architecture that may be used to simulate attacks on a virtual J1939 compliant heavy-duty vehicle [99]. The behavior of the vehicle is replicated using the CANoe environment. The authors of [99] discuss how to extend this solution by linking the simulated environment with physical ECUs, which would enable to evaluate the threat and risk assessment of attacks using the designed controller models running with traffic in the loop. An in-vehicle CAN fuzz-testing message generation model for validating intrusion detection systems was explored by the authors in [100]. Their model is built on a machine learning algorithm, that examines the SAE J1939-specific CAN messages, protocol specifications, and signal correlation to learn about the behavior of the car and then generates the appropriate CAN frames to activate various vehicle functions. The use of powertrace monitoring (the capability of measuring an ECU's power consumption) as a supplementary feedback channel for the fuzz testing applied on embedded systems, was investigated in [101].

2.3 Metrics for evaluating the detection accuracy of an IDS

Since in some of the following chapters several intrusion detection systems are presented, this section provides an overview of the most used performance metrics for the evaluation of an IDS. In the context of an IDS for Controller Area Networks, which is in fact a binary classification between genuine and attack CAN messages, a performance evaluation is based on the following four usual parameters:

1. **True positives (TP)** – stands for the number of attack frames that are correctly classified by the IDS as attacks,
2. **True negatives (TN)** – stands for the number of genuine frames that are correctly classified by the IDS as genuine,
3. **False positives (FP)** – stands for the number of genuine frames that are incorrectly classified by the IDS as attacks,
4. **False negatives (FN)** – stands for the number of attack frames that are incorrectly classified by the IDS as genuine.

Based on these parameters the following metrics are derived:

1. **True positive rate (TPR)** – stands for the ratio between the number of frames that are correctly classified by the IDS as attacks and the total number of attack frames:

$$TPR = \frac{TP}{TP + FN} \tag{2.1}$$

2. **True negative rate (TNR)** – stands for the ratio between the number of frames that are correctly classified by the IDS as genuine and the total number of genuine frames:

$$TNR = \frac{TN}{TN + FP} \tag{2.2}$$

3. **False positive rate (FPR)** – stands for the ratio between the number of frames that are wrongly classified by the IDS as attacks and the total number of genuine frames:

$$FPR = \frac{FP}{FP + TN} \tag{2.3}$$

4. **False negative rate (FNR)** – stands for the ratio between the number of frames that are wrongly classified by the IDS as genuine and the total number of attack frames:

$$FNR = \frac{FN}{FN + TP} \tag{2.4}$$

5. **Accuracy** – is based on all four parameters and stands for the ratio between the number of frames that are correctly classified by the IDS either genuine or attacks and the total number of frames:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.5}$$

6. **Precision** – stands for the ratio between the number of frames that are correctly classified by the IDS as attacks and the number of frames that are classified by the IDS (correctly or wrongly) as attacks:

$$Precision = \frac{TP}{TP + FP} \tag{2.6}$$

Beside these metrics, the validation set **MSE (Mean Squared Error)** is also employed (just for the evaluation of neural network based IDS). This metric measures the average squared difference between the predicted values and the actual ones.

$$MSE = \frac{\sum_{i=0}^{n}(DesiredValue - ActualValue)^2}{n}, \tag{2.7}$$

where n stands for the number of CAN packets from validation set.

Chapter 3

Background on SAE J1939 compliant CAN

This chapter presents the technical details of the Controller Area Network (CAN) protocol. Then, this chapter gives a description of the SAE J1939 standard features and introduces the metrics used for the evaluation of intrusion detection systems.

3.1 CAN bus description

This section gives an overview on general CAN features and then provides some details on the physical layer and data frame formats (standard and extended). In 1983, BOSCH started the development of CAN protocol [14]. The CAN standard, also known as ISO11898, was published ten years later by the International Organization for Standardization (ISO) in 1993. Subsequently, the ISO11898 standard was split in two parts ISO11898-1 [4] and ISO11898-2 [102], respectively. The first part addresses the data-link layer and physical signaling while the second includes details on the high-speed CAN medium access unit. There are two additional ISO11898 that have been published for low speed data communication, i.e., ISO11898-3 [103] and time-triggered communication, i.e., ISO11898-4 [104].

3.1.1 General features

The CAN bus was designed to enable the communication between microcontrollers and devices inside a vehicle. This is as a result of its robustness demonstrated in various severe circumstances such as vibrations produced by cars, atmosphere with a high humidity level as well as high or low temperatures. For low-speed CAN, the data can be transmitted at bit rates of up to 125 $Kbit/s$, while using high-speed CAN, the bit rates can reach 1 $Mbit/s$. A CAN frame can hold a maximum of 8 bytes of data. There are two extensions that were designed in order to support higher speed rate and increased payload. The first one, i.e. CAN with Flexible Data-Rate (CAN-FD) [4] provides bit rates from 2 to 5 $Mbit/s$ while carrying a data of maximum 64 bytes. The second and more recent extension, i.e., CAN Extra Long (CAN-XL) [5] supports data fields of up to 2048 bytes and communication speed of up to 10 $Mbit/s$.

Since CAN is a broadcast protocol, a CAN frame sent by one node can be received by every other node connected to the same network including the node which initiates the transmission. However, if multiple nodes start a message transmission at the same time, the access to the bus is granted to the node with the highest message priority. Outside of the automotive industry, the CAN bus has several applications including building automation [105], 3D printers [106], medical equipment [107], industrial application [108] or avionics [109], [110].

3.1.2 A basic overview on physical layer

Figure 3.1 depicts the topology of a typical CAN network. As shown in the figure, each node is connected to the main bus via a line stub. If the CAN network is responsible for providing the access for diagnostic services an on-board diagnostic port is also connected to the bus in addition to these nodes. There can be two different types of diagnostic ports: the traditional OBD (On-board Diagnostic) port which is typically used in passenger cars, and the J1939 diagnostic port [111], which is specific to heavy-duty vehicles. The CAN physical layer is built on the two differential lines, CAN-High (CAN-H) and CAN-Low (CAN-Low). In order to prevent signal reflections, both sides of the CAN bus must be terminated with a resistor of 120Ω as is shown in Figure 3.1.

The physical and logical high-speed CAN bit format is shown in Figure 3.2. The

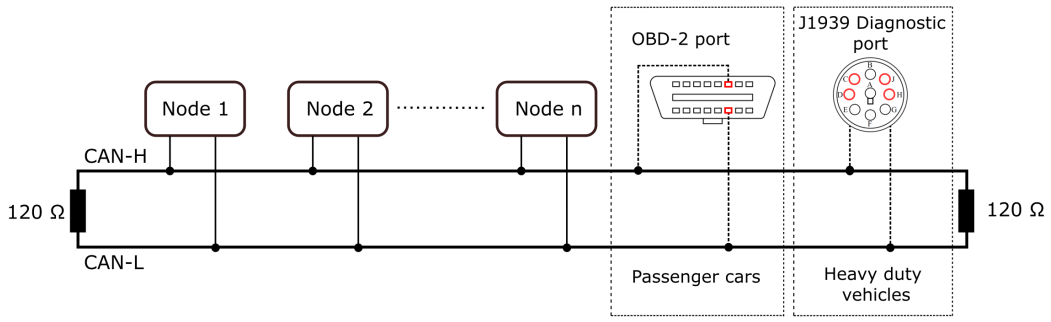


Figure 3.1: Typical CAN network topology for passenger cars and commercial vehicles

transmission of the logical information is encoded using the dominant (logical "0") and recessive (logical "1") states of the physical CAN bit format. The recessive state takes

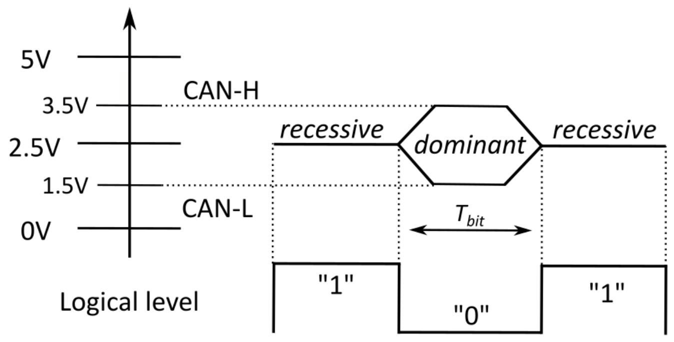


Figure 3.2: Physical representation and logical level encoding for a high-speed CAN bit

place when the signals CAN-H and CAN-L are not actively driven by a network node. During this state, both CAN-H and CAN-L signals exhibit a same voltage which is typically around 2.5 V. In contrast, the dominant state occurs when the bus is driven by at least one network node. When a node is transmitting a dominant bit, the CAN-H signal is about 3.5 V while the CAN-L signal is close to 1.5 V. As a result, the recessive bits are overwritten by the dominant ones. The arbitration and acknowledgment mechanisms, which are detailed in next subsection, are based on the physical layer signaling.

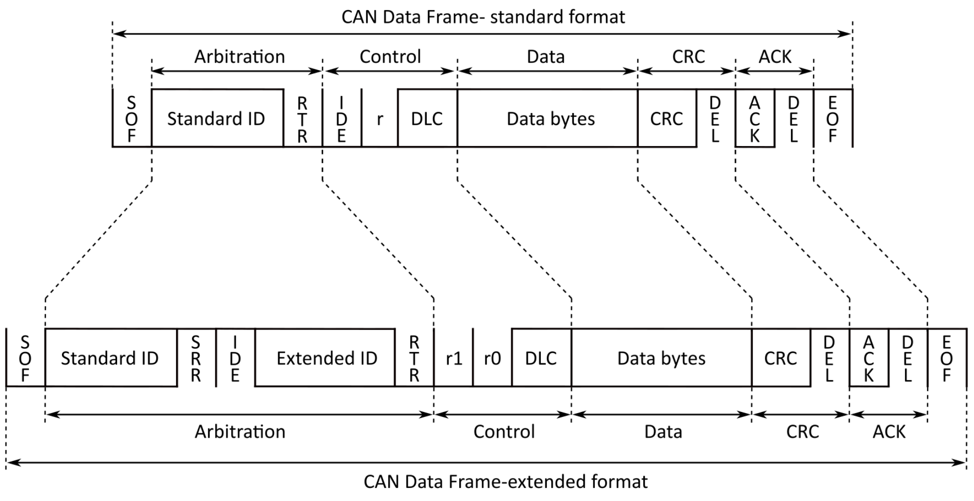


Figure 3.3: Standard and extended CAN data frame structure

3.1.3 The standard and extended formats of CAN data frame

Figure 3.3 depicts the layout of the standard and extended CAN data frame. A dominant Start-of-frame (SOF) bit marks the beginning of each CAN message. The arbitration field, which has a different structure for the standard and extended CAN frame formats, comes after this bit. The arbitration field for the standard format consists of the 11-bit message identifier (ID) and an extra bit called RTR which is "0" (dominant) in case of a data frame or "1" (recessive) in case of a remote frame. On the other hand, the arbitration field includes the 29-bit message ID (11-bit standard ID as well as 18 bit extended ID) in case of extended format. In addition to this ID, the arbitration field accounts for three additional bits. The first two bits, SRR and IDE are placed between standard and extended part of the ID, while the the third bit, i.e., RTR follows after the extended part of the ID. Any CAN message that has a dominant IDE bit is in standard format. The extended format of the CAN message is indicated by a recessive IDE bit. The RTR bit from standard

Table 3.1: CAN2.0B – standard and extended format

Field	Abbr.	Details	Std.	Ext.	Length (bits)
Start-of frame	SOF	dominant bit that marks the beginning of the message transmission	✓	✓	1
Base Identifier	Base ID	unique identifier for a message considering the standard format, base ID (first part) for a message considering the extended format	✓	✓	11
Substitute remote request	SRR	recessive bit	–	✓	1
Identifier extension bit	IDE	dominant for standard frames while recessive for extended frames	✓	✓	1
Extended Identifier	Ext. ID	extended ID (second part) for the extended frames	–	✓	18
Remote transmission request	RTR	recessive for a remote request message, dominant in case of a data message	✓	✓	1
Reserved bit	r1	must be set as dominant by the transmitter, but accepted in any state by the receivers	–	✓	1
Reserved bit	r0	must be set as dominant by the transmitter, but accepted in any state by the receivers	✓	✓	1
Data length code	DLC	the number of data bytes carried by the frame	✓	✓	4
Data field	-	the data that is packed inside the frame	✓	✓	0-64
Cyclic redundancy check	CRC	a 15-bit checksum sent by the transmitter for data integrity checks	✓	✓	15
Cyclic redundancy check delimiter	CRC del.	recessive bit	✓	✓	1
Acknowledge bit	ACK	set as recessive by the frame transmitter and set to dominant by frame receivers	✓	✓	1
Acknowledge delimiter	ACK del.	recessive bit	✓	✓	1
End-of-frame	EOF	recessive bits	✓	✓	7

format is replaced in extended format by the SRR bit, which is always sent as recessive to ensure that a standard data frame will win the arbitration with an extended data frame, in case both frames share the same 11-bit base identifier. After the arbitration field there is the control field, which accounts for one reserved bit for standard format or two reserved bits for extended one and DLC. The DLC specifies how many bytes (0-8) are packed and sent as payload (data field). To ensure the data integrity, an error detecting code, i.e., a 15-bit CRC, is computed over the previously transmitted bits from the frame and is placed inside the CRC field. The ACK field starts with a ACK bit and ends with a recessive ACK delimiter bit. The transmitters set the ACK bit to be recessive, and each node that successfully receives the frame overwrites the recessive bit with a dominant bit to confirm the reception of the CAN frame. The EOF field, which contains 7 recessive bits, marks that the CAN message has ended. A detailed overview on the format and bit fields of the standard and extended CAN frame is shown in Table 3.1.

3.2 SAE J1939 specifics

In this section, J1939 specific implementations such as the message identifier (ID) structure, the address claiming mechanism as well as the multi-frame transmissions are presented. The SAE J1939 specification for the CAN protocol is used by the commercial vehicle sector of the automotive industry. Any vehicle that is employed in commercial or business purposes, e.g., deliver goods or to transport passengers for a fee, is qualified as a commercial vehicle. This specific sector comprises various motor vehicles including buses, trucks, mobile cranes, excavators, tractors etc.

With a few minor modifications, i.e., bit rate restrictions, SAE J1939 implements the ISO 11998 standard specification at the physical layer. A SAE J1939 vehicle network has a maximum bit rate of up to 500 $Kbit/s$. When developing CAN buses inside commercial vehicles, the typical employed bit rates are 250 $Kbit/s$ or 500 $Kbit/s$. On top of the CAN protocol, SAE J1939 defines a full communication stack by including additional specific features, such as the data link standardized in SAE J1939-21 [112] and the vehicle application layer specified in the SAE J1939-71 standard [81].

3.2.1 ID breakdown based on J1939 standardization

Only extended frames, meaning frames that have 29-bit identifiers, are used in J1939 CAN communication buses. This is a specific feature of J1939 standard, while regular CAN buses used inside passenger cars typically follows the standard format, which consists of frames with 11-bit identifiers. Figure 3.4 depicts the specific layout designed for the J1939 specific ID. The J1939 frame ID is split into six separate fields as follows: priority, extended data page (EDP), data page (DP), protocol data unit format (PF), protocol data unit specific (PS) and source address (SA). The Parameter Group Number (PGN) uniquely

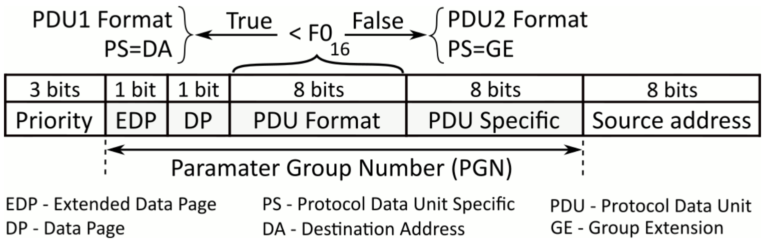


Figure 3.4: J1939 message identifier layout

identifies a parameter group (PG) which specifies that a set of particular parameters, such as data, acknowledgments, etc., are packed inside the payload of a J1939 frame. The following four fields: EDP, DP, PF, PS defines the PGN. The J1939-71 standard [81] describes a complete set of standardized J1939 frames and the associated PG values, which meet the requirements of various functions that are implemented on ECUs inside the commercial vehicles. In addition to the set of J1939 standard frames, there is also defined a range that is reserved for the OEM specific implementations [81]. As long as the particular frame type is not OEM-specific, the data field carried by a J1939 frame (the parameters that are packed inside the payload) can be interpreted based on J1939 specifications. The J1939-71 standard is supplemented by the J1939 Digital Annex document [113], which gives a detailed information in a format that is easy to use. Additionally, all the J1939 targeted areas, such as agricultural and forestry equipment, on-highway equipment, etc. and the corresponding source addresses for different ECUs preferred by each area are listed in this document. The J1939 specification defines two types of PDU formats. This classification is performed based on the PDU format value following this rule:

1. if the value of the PF field is less than 240, the PDU format is labeled as PDU1.
2. if the value of the PF field is greater than or equal to 240, the PDU format is labeled as PDU2.

The significance of PS field is the main difference between the two aforementioned types of the PDU format. In case of a PDU1 format, the PS field specifies a destination address while for the PDU2 means a group extension (GE).

3.2.2 Address claiming mechanism based on node address

A node address is an unique identification number for a J1939 network node. These addresses are a component of the message identifiers and represent the source and destination of the frames. The DA will be `0xFF` in the case of a broadcast transmission which means a transmission that is addressed to all nodes from the network. Another important aspect is that the only nodes with allocated addresses are allowed to exchange messages.

The address claiming mechanism, which occurs prior to the start of the communication inside the J1939 network, allows the nodes to claim an address. This procedure is part of the J1939 network management specification described in SAE J1939-81 [114] and occurs while the nodes are waking up, typically at system ignition phase. There are some circumstances when the nodes are initialized on request or are later connected to the network. In this scenario, the nodes will go through the identical address claiming process, which the other nodes have already performed. According to [114], the CAN message that has an ID with PGN 60928, corresponds to the address claiming procedure. The transmission of this PGN is performed on request. As a result, any network node may send this request (a particular frame with PGN 59904). The destination of such a request can be either a global address, i.e., all nodes, or a particular address of a node. Before starting the communication, a node must send a frame with PGN 60928, containing its NAME and the requested address in order to claim its own address. In case of a global request, each ECU within the J1939 network must reply with a frame based on the Address Claiming PGN 60928 providing its specific NAME and address if the node has previously claimed an address. A node which is unable to claim an address must notify this failure by including the null address (254) in the address field.

Each node has assigned a unique NAME parameter which provides details on the ECU's functions, the manufacturer code, the identify number and the industry sector. Figure 3.5 is a depiction of the address claiming mechanism. As it can be observed, the ECU1 designated as the *Initiator node* claims his own address in the first phase. The ECU1 node then uses the global address to send a message to all other nodes that is a PGN request for an Address Claim. All J1939 network ECUs that have already claimed an address respond by sending an Address Claim message with their own source address.

3.3 Data transmissions via multi-frame

Transport protocols, which are another feature of the J939 communication standard, are described in [112]. A J1939 CAN network uses transport protocols that enable the transmission of larger payloads, i.e., up to 1785 bytes, by using multi-frame messages. Since a CAN message can pack maximum 8 data bytes, messages that need to carry more than 8 bytes may only be transmitted by splitting the data across multiple CAN frames. This is feasible due to the existence of various J1939 specific frames, which are detailed in Table 3.2. These frames are part of both the Transport Protocol Connection Management (TP.CM) and the Transport Protocol Data Transfer (TP.DT). A connection between the sender and receiver nodes is established via a TP.CM message, which also manages the frame exchange. The actual data transfer is done using the TP.DT messages. Both, the TP.CM and TP.DT messages enables two types of data transfer as follows:

1. **Broadcast Data Transfer** – as illustrated in Figure 3.6 (left). Here, the initiator node labeled as ECU1 transmits a Broadcast Announce Message (BAM) in the first

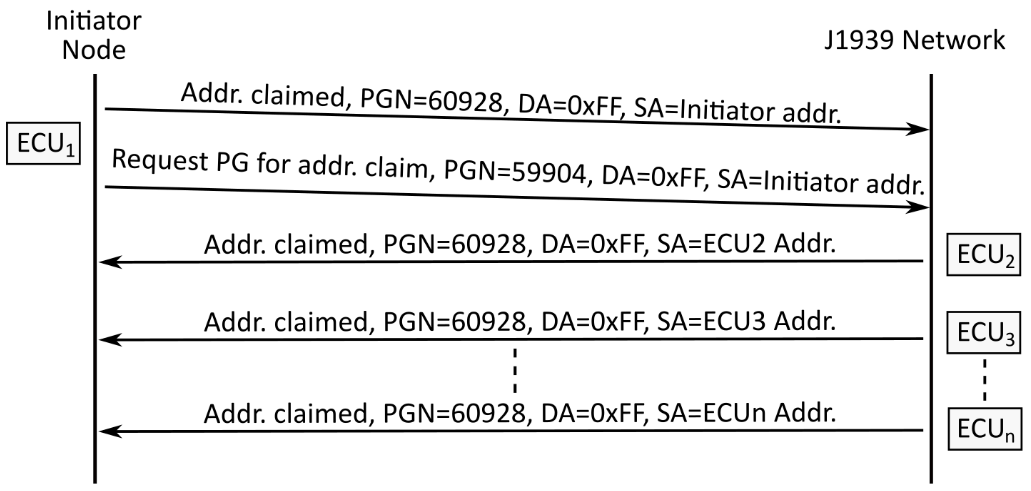

Initiator Node
J1939 Network
ECU1
Addr. claimed, PGN=60928, DA=0xFF, SA=Initiator addr.
Request PG for addr. claim, PGN=59904, DA=0xFF, SA=Initiator addr.
Addr. claimed, PGN=60928, DA=0xFF, SA=ECU2 Addr.
ECU2
Addr. claimed, PGN=60928, DA=0xFF, SA=ECU3 Addr.
ECU3
Addr. claimed, PGN=60928, DA=0xFF, SA=ECUn Addr.
ECUn

Figure 3.5: The flow of the messages exchanged while the J1939 address claiming mechanism takes place

step to let the network know that a multi-packet message is coming. The DT (Data Transfer) frames are then sent by the same node. Only the initiator node has control over the message exchange.

2. **Connection Mode Data Transfer** – as illustrated in Figure 3.6 (right). In this scenario, an RTS (Request to Send) - CTS (Clear to Send) message exchange is used to create a connection between the sender and receiver nodes. Next, the DT messages are sent by the initiator node (ECU1). Finally, the receiver node sends an End of Message Acknowledgment (EndOfMsgACK) message to confirm the end of a multi-frame transmission. The Connection Abort (Conn Abort) message can be used at any time to suspend the connection between the initiator and receiver nodes.

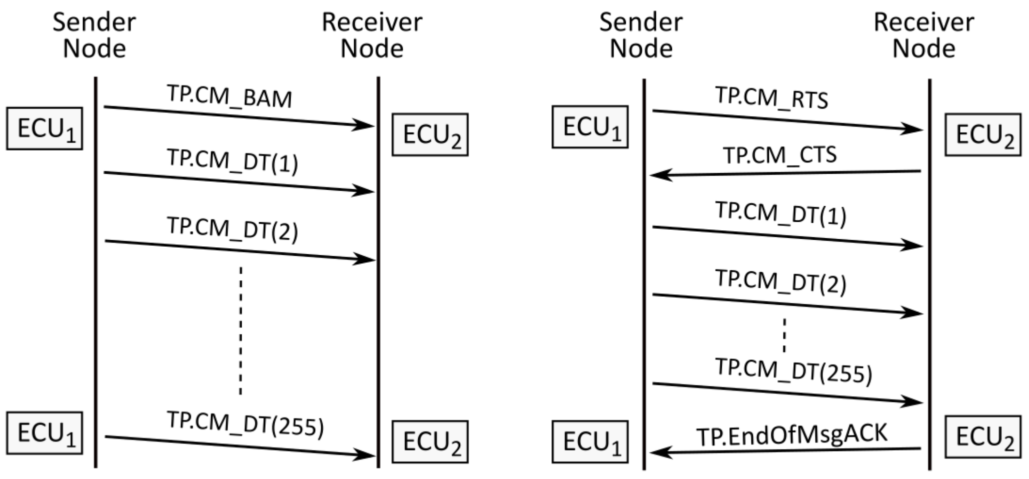


Figure 3.6: Broadcast Data Transfer (left) and Connection Mode Data Transfer (right) – transport protocols used in multi-packet transmissions

Table 3.2: Transport protocol messages used in J1939 CAN networks

Message	PF	PS	Description
TP.CM_BAM	236	FF	through this message, the nodes are notified about a specific PG and the number of data bytes that will be packed into a multi-frame message.
TP.CM_RTS	236	DA	a node can inform another node of their intention to connect with it for a multi-frame message transmission.
TP.CM_CTS	236	DA	the receiver transmits this frame as response to a TP.CM RTS frame to let the connection initiator node know that is available to receive the specified amount of data bytes from the RTS message.
TP.CM_EndOfMsgACK	236	DA	through this message, the receiver node notifies the sender node that all data bytes have been successfully received
TP.Conn_Abort	236	DA	both the receiver or sender can use this message to suspend a connection before data transmission is completed

Chapter 4

A preliminary approach with a neural network based IDS on CAN

This chapter explores the effectiveness of neural networks in detecting intrusions in Controller Area Networks. Also, it provides computational results for the detection algorithm on three automotive graded platforms. The experiments are conducted on a CAN trace that is taken from a J1939 CANoe simulation as well as on a publicly accessible CAN dataset. In addition to the advantages of utilizing neural networks for intrusion detection, this chapter also discusses the limitations caused by the required computational time which is essential for real-time detection. This chapter is based on the results published in a prior work of the author [22]. These experiments were started while the author was enrolled on the MSc program and the current results are an extension of those included in the author's dissertation thesis [115]. The detection results were evaluated using the MATLAB Neural Network Toolbox and the C++ implementation of the detection algorithm was used for measuring the runtime on different embedded platforms.

4.1 Targeted scenario and adversary model

This section outlines the scenario that an adversary might employ as well as the types of attacks that are examined in this evaluation.

Figure 4.1 depicts the targeted setup which accounts for an adversary that is connected to the CAN bus through the OBD port by using some malicious device and injects messages on the bus. In this chapter, the CANoe trace is augmented with adversarial frames using a C# based script. The following types of attacks are considered:

- *Fuzzing attacks* – are attacks in which the adversary injects malicious CAN messages that have the same ID as a regular frame and the content of the data field is randomly

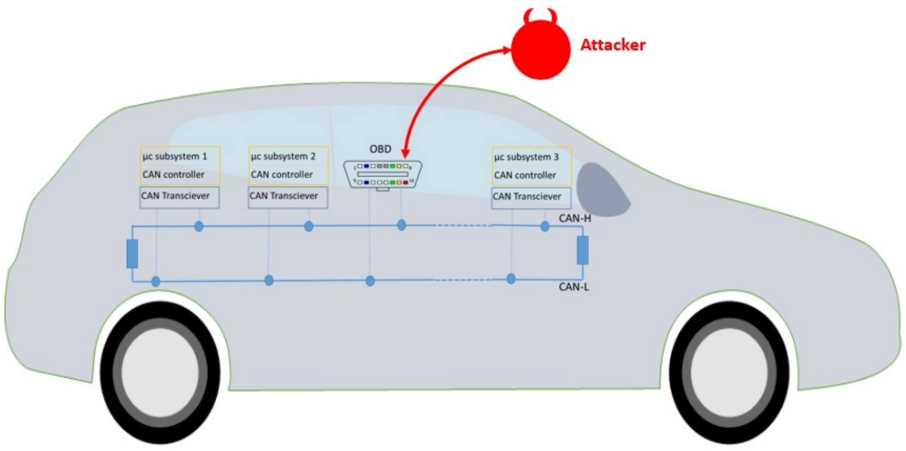


Figure 4.1: Targeted setup

generated. The timestamp of the attack frame is randomly generated between the timestamps of two consecutive genuine CAN messages (we assume that this adversarial behavior matches with real-world attack scenarios). As a result, the attack frame occurs at a random location in the legitimate trace. When a certain CAN ID is targeted by an adversary, the intrusions take place randomly between the timestamps of two consecutive genuine messages with the targeted ID. When the full trace is considered, i.e, all CAN IDs, the intrusions occur at random points, no later than several dozen messages from the legitimate one.

- *Replay of regular CAN frames* – are attacks in which the adversary injects malicious CAN messages that have the same ID and data field as a genuine frame. The attack frames take place at random locations, identical to the case of fuzzing attacks.

The proposed adversary model is comparable to the one used in [116]. The replay and fuzzing attacks that are described in [116] are the same as the ones in the current work. Apart from that, the authors in [116] also take into account DoS (Denial of Service) attacks that are mounted by sending frames on the CAN bus with ID 0x000h, i.e., the highest priority. However, these attacks are trivial to spot because ID 0x000h is not part of the legitimate traffic. As a result, this type of attack is neglected in the evaluation from the current chapter because its detection is straightforward.

4.2 Neural networks deployment and tools

This section presents the tools used for the deployment of neural networks and provides a basic explanation of their architecture.

4.2.1 Neural network Toolbox and C++ implementation

For the experiments that follow, both the Neural Network Toolbox provided by an industry standard tool, i.e., MATLAB and an independent C++ implementation, are used. The rationale behind employing both of these implementations is that the MATLAB neural network toolset is well-known for its performance and features. Nevertheless, an open-source C++ code is more suitable for deployment on embedded platforms. Due to this, the majority of the detection results were obtained using MATLAB. However, we verify that comparable detection results are achieved by using separate C++ code. [1]. Consequently, this C++ implementation is used for benchmarking the neural network based detection algorithm on automotive graded platforms.

The MATLAB toolbox includes several algorithms to address classification problems. To be more specific, for the current evaluation the training is done using the *trainscg* algorithm, which is the scaled conjugate gradient back-propagation. This algorithm updates the weights and bias values. The hyperbolic tangent sigmoid transfer function, i.e, *tansig*, is used for connecting the layers of the neural network. This function provides a value between $[-1, 1]$. The reason for choosing the *tansig* function is that, according to the MATLAB documentation, it provides good trade-offs where computational speed is crucial. The back-propagation method and the sigmoid function are also used in the C++ implementation.

To facilitate the training of the neural network and validating the results, the CAN traffic dataset is split into three parts:

- *training data (TD)* – the data required for training the network, i.e., adjusting the weights of the network accordingly,
- *validation data (VD)* – the data required to evaluate how well the neural network performs when encountering new data (every epoch ends with the running of this input),
- *test data (TsD)* – the data required after the training stage is finished (when the stop conditions are met), and the accuracy of the detection results are computed over this data.

We note that the test data from MATLAB implementation is referred to as validation data in the C++ implementation while the validation data is referred to as generalization data (this is just a naming convention).

An epoch represents the time in which both the entire training data set, as well as the generalization data, are passed through the neural network which continues to operate until the stopping criteria are satisfied. These are detailed next for MATLAB implementation. As the training stage only took a few minutes or less, and since the training stage is

[1]https://takinginitiative.wordpress.com/2008/04/23/basic-neural-network-tutorial-c-implementation-and-source-code/

an off-line procedure that doesn't need to satisfy real-time requirements, the *maximum amount of time*, one of the stopping criteria for the training, was set to the default setting, i.e, ∞. This permits stopping on one of the subsequent criteria:

- the *maximum epoch reached* (configured to 1000),
- the *performance goal* is achieved (configured to 0),
- the *performance gradient* is lower than the minimum gradient (configured to 1e-06),
- the *validation performance* has improved for at least six consecutive times.

Similar stopping criteria were present in the C++ implementation. For instance, as long as the accuracy of both the training and generalization set is below the required accuracy, the network will continue to operate until the maximum amount of epochs is achieved. If the previous stop conditions are not met, the *training set accuracy* (TSA), which is the number of CAN frames accurately predicted as either attacks or legitimates frames, can be used as a condition. The *training set accuracy* (TSA) is computed as follows:

$$TSA = 100\left(1 - \frac{NIC}{NT}\right) \tag{4.1}$$

where *NIC* stands for the number of incorrect predictions and *NT* represents all of the CAN messages from the training data. The last stopping criterion is the *generalization set accuracy* (GSA), which is the same as TSA, but is computed using the generalization data.

4.2.2 Neural network architecture

Figure 4.2 shows the neural network architecture which consists of three layers: the input layer, the hidden layer, and the output layer. The neural network input contains the following features extracted from the CAN frames:

- the data field,
- the frame ID which has 29 bits in case of extended CAN frame,
- the interval (Δt) between timestamps with the same ID that are consecutive.

The data input vector $I \in \{0,1\}^{105}$ can be mathematically defined as:

$$I = \left\{b_0, b_1, b_2, ...b_{104}\right\} \text{ where:} \tag{4.2}$$

bits $b_0...b_{11}$ stand for the interval Δt, bits $b_{12}...b_{75}$ stand for the 64-bit data field, and bits $b_{76}...b_{104}$ stand for the 29-bit extended ID.

The output $O \in \{0,1\}$ is defined as:

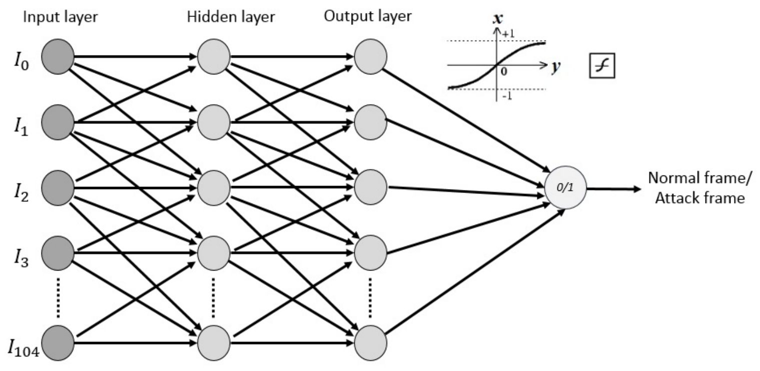


Figure 4.2: Neural network architecture

$$O = \{b_0\} \text{ where:} \tag{4.3}$$

$$b_0 = \begin{cases} 1, \text{denotes an attack message} \\ 0, \text{denotes a genuine message} \end{cases} \tag{4.4}$$

4.3 Experimental results

In this section, the detection results obtained for various experiments as well as the computational results of the detection algorithm on several embedded platforms, are presented.

4.3.1 Results on detecting intrusions using neural networks

The proposed IDS is evaluated using a trace that was recorded from a J1939 CANoe simulation. In the first step, the performance evaluation is conducted on portions of traces containing all frames with a certain CAN identifier. For the experiments that follow three rates of injection are used: 5%, 10% and 20%. For each injection rate there are five cases of data separation between three categories: training (TD), validation (VD) and testing (TsD). These are presented in Table 4.1.

The first experiment was performed on a public real-world CAN dataset made accessible by the authors in [116]. The traces from the dataset contains genuine CAN traffic as well as CAN traffic that was augmented with malicious frames. Unfortunately, for the later scenario, there is no indicator in the traces to distinguish between malicious frames and legitimate frames. As a result, only traces containing fuzzing attacks were used in the

Table 4.1: The separation of the dataset for each injection rate based on the training, validation, and testing data

I	60% TD	20% VD	20% TsD
II	40% TD	20% VD	40% TsD
III	20% TD	20% VD	60% TsD
IV	10% TD	10% VD	80% TsD
V	5% TD	5% VD	90% TsD

Table 4.2: Detection results on fuzzing attacks from the public dataset

Inj.	Case	Neural Network Parameters				Results			
		Validation set max fail	Gradient	Validation set MSE	Nr. epochs	TN & TNR	TP & TPR	FP & FPR	FN & FNR
N/A	I	1	7.07e-07	9.6381e-07	24	2454 99.92%	5586 100%	2 0.08%	0 0%
N/A	II	1	5.21e-07	7.5294e-04	31	5277 99.96%	10806 100%	2 0.04%	0 0%
N/A	III	1	8.22e-07	7.4193e-07	28	7860 99.97%	16266 100%	2 0.03%	0 0%
N/A	IV	1	6.08e-07	1.4023e-06	27	10462 99.98%	21707 100%	2 0.02%	0 0%
N/A	V	1	6.49e-07	1.4732e-06	26	11827 99.98%	24363 100%	2 0.02%	0 0%

experiments, with the assumption that the randomized frames are attacks while the other frames are legitimate. The results are excellent, with a detection rate equal to 100% while the false positive rates are below 0.1%, as shown in Table 4.2. This is primarily because the adversary behavior is straightforward, i.e., it only targets a single ID. To explore the limits of the neural network-based detection, the upcoming experiments will be more complicated.

Results obtained for a single ID. In what follows, the detection performance is evaluated on a single CAN ID from the J1939 CANoe trace. The detection results on a single ID are relevant as a baseline because it is obvious that expanding detection to include all of the IDs from the trace will require a neural network for each CAN ID, in other words, it requires powerful resources in terms of processing power and storage capacity. It is worth mentioning that the CAN traffic includes low entropy frames carried by some IDs, i.e., roughly 0, and high entropy frames by other IDs, i.e., 12-13 bits of entropy. This is due to the fact that the data fields are mostly constant for the former while those for the latter carry parameters that exhibit variations. Consequently, the evaluation is conducted to

Table 4.3: Detection results for injections with random data obtained using a low-entropy frame

Inj.	Case	Neural Network Parameters				Results			
		Validation set max fail	Gradient	Validation set MSE	Nr. epochs	TN & TNR	TP & TPR	FP & FPR	FN & FNR
20%	I	0	8.79e-07	1.0677e-07	29	7519 100%	1518 100%	0 0%	0 0%
20%	II	0	9.23e-07	5.7362e-07	27	15079 100%	2995 100%	0 0%	0 0%
20%	III	1	7.21e-07	1.7136e-06	26	22645 100%	4466 100%	0 0%	0 0%
20%	IV	0	7.34e-07	1.1782e-06	31	30152 100%	5996 100%	0 0%	0 0%
20%	V	1	8.12e-07	7.9371e-05	25	33871 100%	6794 99.99%	0 0%	1 0.01%

account for both scenarios, i.e, low and high entropy frames. For injections with random data (fuzzing attacks), the results are shown in Table 4.3 for a frame ID with low entropy and Table 4.4 for a frame ID that carries a high entropy. Although there is a minor increase in the false-negative rate for the frame with higher entropy, the detection rate is still very close to 100% while none of the legitimate frames is classified as an intrusion.

For the case of replay attacks, the results are presented in Tables 4.5 and 4.6. In contrast with fuzzing attacks, the low-entropy frame exhibits slightly worse detection results. The detection rate for low-entropy frame can occasionally drop to around 86%, and it is over 97% for the high-entropy frame. The true negative rate remains constant at 100% for both scenarios.

Results obtained over the full trace. In what follows, the evaluation of the detection performance is conducted using the full trace. The results for both fuzzing and replay attacks are shown in Tables 4.7 and 4.8, respectively. The attack traces were built in a similar vein to those targeting a single ID, but to cover all IDs from the trace. The false positive rate has increased for both types of attacks as a result of the extension over the full trace. Nonetheless, the false positive rates remain within a few percentage points (0%-2%), and only in the case of replay attacks with an injection rate of 20%, they reach 11.09% and 13.83%. A significant concern is the false negative rate, which shows a noticeable increase at roughly 43.58% in case of replay attacks with an injection rate of 5%. Since the false positive rate decreases from 43.58% to 17.64% when the injection rate increases from 5% to 20%, the reduced number of attack frames (5% injection rate) from the training set may be the cause for the higher rates of false positives.

Results obtained over the full trace using a reduced network size. Considering that the computational power and storage demands may be too high for automotive-graded

Table 4.4: Detection results for injections with random data obtained using a high-entropy frame

Inj.	Case	Neural Network Parameters				Results			
		Validation set max fail	Gradient	Validation set MSE	Nr. epochs	TN & TNR	TP & TPR	FP & FPR	FN & FNR
20%	I	1	9.63e-07	3.7269e-07	28	7562 100%	1475 100%	0 0%	0 0%
20%	II	1	9.57e-07	7.5004e-06	28	15093 100%	2981 100%	0 0%	0 0%
20%	III	1	6.32e-07	3.281e-05	28	22645 100%	4457 100%	0 0%	0 0%
20%	IV	0	8.27e-07	3.8664e-05	28	30161 100%	5987 100%	0 0%	0 0%
20%	V	0	7.31e-07	5.3963e-06	29	33890 100%	6774 99.97%	0 0%	2 0.03%

Table 4.5: Detection results for replay attacks obtained using a low-entropy frame

Inj.	Case	Neural Network Parameters				Results			
		Validation set max fail	Gradient	Validation set MSE	Nr. epochs	TN & TNR	TP & TPR	FP & FPR	FN & FNR
20%	I	6	1.10e-03	2.9128e-04	26	7530 100%	1507 100%	0 0%	0 0%
20%	II	6	2.32e-03	6.1636e-06	37	15086 100%	2909 97.36%	0 0%	79 2.64%
20%	III	6	6.64e-04	6.0327e-03	33	22601 100%	4297 95.28%	0 0%	213 4.72%
20%	IV	6	2.74e-03	4.3834e-03	16	30081 100%	5639 92.95%	0 0%	428 7.05%
20%	V	6	4.23e-03	2.5314e-02	13	33886 100%	5855 86.36%	0 0%	925 13.64%

Table 4.6: Detection results for replay attacks obtained using a high-entropy frame

Inj.	Case	Neural Network Parameters				Results			
		Validation set max fail	Gradient	Validation set MSE	Nr. epochs	TN & TNR	TP & TPR	FP & FPR	FN & FNR
20%	I	6	9.61e-07	2.0248e-06	43	7557 100%	1449 97.91%	0 0%	31 2.09%
20%	II	1	6.70e-07	3.7639e-03	45	15051 100%	2992 98.97%	0 0%	31 1.03%
20%	III	6	6.64e-04	6.0327e-03	33	22592 100%	4431 98.05%	0 0%	88 1.95%
20%	IV	1	8.85e-07	4.2954e-05	43	30147 100%	5936 98.92%	0 0%	65 1.08%
20%	V	0	5.92e-07	1.8005e-05	42	33886 100%	6585 97.12%	0 0%	195 2.88%

Table 4.7: Detection results for injections with random data obtained using the full trace

Inj.	Case	Neural Network Parameters				Results			
		Validation set max fail	Gradient	Validation set MSE	Nr. epochs	TN & TNR	TP & TPR	FP & FPR	FN & FNR
5%	IV	1	9.15e-07	9.4955e-04	47	79842 99.68%	3833 98.26%	257 0.32%	68 1.74%
5%	V	1	7.51e-07	8.2342e-04	50	89782 99.70%	4368 98.18%	269 0.3%	81 1.82%
20%	IV	0	6.07e-07	1.6087e-03	48	80137 99.64%	15552 99.85%	288 0.36%	23 0.15%
20%	V	1	8.34e-07	3.6169e-04	41	89850 99.62%	17700 99.42%	347 0.38%	103 0.58%

Table 4.8: Detection results for replay attacks obtained using the full trace

Inj.	Case	Neural Network Parameters				Results			
		Validation set max fail	Gradient	Validation set MSE	Nr. epochs	TN & TNR	TP & TPR	FP & FPR	FN & FNR
5%	IV	6	1.41e-02	1.8857e-02	213	78749 98.37%	2445 61.96%	1305 1.63%	1501 38.04%
5%	V	6	3.47e-03	2.1721e-02	167	88401 98.20%	2526 56.42%	1622 1.80%	1951 43.58%
20%	IV	6	2.01e-02	3.8511e-02	150	69220 86.17%	12909 83.26%	11106 13.83%	2765 17.64%
20%	V	6	9.12e-03	3.7179e-02	158	80157 88.91%	14430 80.88%	10001 11.09%	3412 19.12%

Table 4.9: Detection results for injections with random data obtained using the full trace and 1/4 hidden layer size

Inj.	Case	Neural Network Parameters				Results			
		Validation set max fail	Gradient	Validation set MSE	Nr. epochs	TN & TNR	TP & TPR	FP & FPR	FN & FNR
5%	IV	0	9.35e-07	4.9171e-04	45	79842 99.68%	3846 98.28%	257 0.32%	67 1.72%
5%	V	1	4.60e-07	2.0366e-03	46	89665 99.57%	4275 96.09%	386 0.43%	174 3.91%
20%	IV	6	5.42e-07	7.3191e-04	52	80153 99.66%	15462 99.27%	272 0.34%	113 0.73%
20%	V	1	9.90e-07	1.0166e-03	50	89471 99.20%	17695 99.39%	726 0.80%	108 0.61%

controllers, as demonstrated in the next section, reducing the network size is essential. The detection results for injections with random data and replay attacks obtained using a network that has a hidden layer reduced to 1/4 are presented in Tables 4.9 and 4.10. Similarly, the detection results obtained for the same attack scenarios, but based on a network that has a hidden layer reduced to 1/16 are shown in Tables 4.11 and 4.12, respectively. As anticipated, the accuracy performance is affected as network size is reduced. Nevertheless, the detection results obtained with both 1/4 and 1/16 size of the hidden layer, are acceptable. Once more, the primary concern is the false negative rate in the case of replay attacks with an injection rate of 5%, which is around 40% and around 55% when the hidden layer is reduced to 1/4 and 1/16, respectively. The detection results show an improvement after the injection rate is increased to 20%, indicating the necessity for a larger training set.

Table 4.10: Detection results for replay attacks obtained using the full trace and 1/4 hidden layer size

Inj.	Case	Neural Network Parameters				Results			
		Validation set max fail	Gradient	Validation set MSE	Nr. epochs	TN & TNR	TP & TPR	FP & FPR	FN & FNR
5%	IV	6	8.93e-03	1.8363e-02	281	78719 98.33%	2476 62.75%	1335 1.67%	1470 37.25%
5%	V	6	1.07e-02	1.8056e-02	214	89050 98.92%	2711 60.55%	973 1.08%	1766 39.45%
20%	IV	6	1.82e-02	3.4832e-02	306	72010 89.65%	12742 81.29%	8316 10.35%	2932 18.71%
20%	V	6	1.09e-02	3.4798e-02	154	84019 93.19%	14046 78.72%	6139 6.81%	3796 21.28%

Table 4.11: Detection results for injections with random data obtained using the full trace and 1/16 hidden layer size

Inj.	Case	Neural Network Parameters				Results			
		Validation set max fail	Gradient	Validation set MSE	Nr. epochs	TN & TNR	TP & TPR	FP & FPR	FN & FNR
5%	IV	6	1.79e-04	1.1984e-03	53	79818 99.65%	3757 96.31%	281 0.35%	144 3.69%
5%	V	6	3.15e-05	1.6704e-03	39	89444 99.33%	4229 95.06%	607 0.67%	220 4.94%
20%	IV	6	4.0e-05	1.1148e-03	42	80007 99.48%	15401 98.88%	418 0.52%	174 1.12%
20%	V	6	7.50e-05	2.3672e-03	54	88763 98.41%	17439 97.96%	1434 1.59%	364 2.04%

Table 4.12: Detection results for replay attacks obtained using the full trace and 1/16 hidden layer size

Inj.	Case	Neural Network Parameters				Results			
		Validation set max fail	Gradient	Validation set MSE	Nr. epochs	TN & TNR	TP & TPR	FP & FPR	FN & FNR
5%	IV	6	7.86e-03	2.8314e-02	125	79528 99.34%	1791 45.39%	526 0.66%	2155 54.61%
5%	V	6	4.45e-03	2.6876e-02	113	88184 97.96%	1871 41.79%	1839 2.04%	2606 58.21%
20%	IV	6	1.35e-02	4.1603e-02	189	63747 79.36%	12704 81.05%	16579 20.64%	2970 18.95%
20%	V	6	1.69e-02	4.5909e-02	202	74453 82.58%	13068 73.24%	15705 17.42%	4774 26.76%

4.3.2 Runtime performance for neural networks

The computational power and storage capacity of the automotive embedded platforms used in the implementation of the IDS are essential for the effective deployment in real-world automotive applications. To assess the computational performance of the proposed IDS, the experiments are conducted to measure the runtime of the detection mechanism on three automotive-grade platforms. A microcontroller from NXP, i.e., S12XF512, is the first and represents the low-end device sector. As representatives of the high-end device group, an Infineon AURIX TC297 microcontroller as well as a Renesas RH850/E1x are used. The S12XF chip is equipped with a 16-bit main core that can run at a maximum frequency of 50MHz, 32KB of RAM as well as 512KB of Flash. The Infineon AURIX TC297 microcontroller, on the other hand, offers three 32-bit cores that can operate at up to 300MHz along with 728KB of RAM and 8MB of Flash. For the second device, i.e, Renesas RH850/E1x the evaluation was done utilizing a dedicated simulator. Two 32-bit cores running at a maximum clock speed of 320MHz along with 352KB of RAM and 4MB of Flash are available on the Renesas platform.

The first step is to compute the weights in an off-line manner based on the training data set. The second one is the testing stage where the neural network based detection algorithm was designed to operate on a single-core, while the precomputed weights are stored in Flash memory. The experiments were carried to use three different sets of weights that represent the neural network with a full size hidden layer, a hidden layer reduced to 1/4 as well as a hidden layer reduced to 1/16. For each set of weights, the testing was conducted using two floating point formats, i.e, single and double precision. The runtimes determined for the proposed IDS in the investigated scenarios are presented in Tables 4.13 and 4.14. Due to a constraint in the compiler that we used, the code could not be executed on the S12XF device with the full size hidden layer.

Table 4.13: Computational results obtained using single precision floats for weights

Platform	Full hidden layer	1/4 hidden layer	1/16 hidden layer
S12XF512	n/a	$52.3ms$	$13.22ms$
TC297	$3.904ms$	$899\mu s$	$237.5\mu s$
RH850/E1x-FCC1	$2.697ms$	$667.1\mu s$	$157.6\mu s$

Table 4.14: Computational results obtained using double precision floats for weights

Platform	Full hidden layer	1/4 hidden layer	1/16 hidden layer
S12XF512	n/a	$110.5ms$	$25.76ms$
TC297	$15.26ms$	$3.744ms$	$822\mu s$
RH850/E1x-FCC1	$2.680ms$	$671.3\mu s$	$162.73\mu s$

According to the expectations, the low-end platform exhibits a significant performance bottleneck, making the implementation of the detection algorithm only possible for a limited neural network size and a CAN network with a reduced number of nodes that sends frames with high periodicity (i.e. greater than 100ms). The results of the experiments demonstrate that the detection algorithm can be implemented more effectively on high performance devices. However, real-time processing of the CAN frames with a periodicity in the order of 10 milliseconds may still be a concern when implementing the full version of the suggested neural network.

4.4 Concluding remarks

Although neural networks show promising capabilities in detecting CAN bus intrusions, there are still certain limitations. The obtained results, as expected, differ between the two types of attacks, replay and injections with random data. Due to the fact that in case of replay attacks the injected frames are identical to the legitimate ones, the IDS exhibits a lower detection rate for such attacks. The only indicator to detect such attacks is the frequency of the CAN frames. On the other hand, the injections with random data are easily detected by the IDS.

Neural networks appear to be an effective IDS for CAN buses from the perspective of detection results. However, the bigger concern is with the computational and storage requirements since neural networks don't seem to be appropriate for real-time detection on low-end automotive-grade controllers. While reduced-size neural networks may be supported by high-end controllers, computational needs remain considerable. Consequently, unless specialized hardware becomes available, the detection phase may not always be performed locally on each ECU from the network. A potential solution could involve relying on gateways with more powerful cores that can monitor the traffic while adhering to time constraints. The investigation of such a solution with stronger cores and the extension of this evaluation to include more intricate real-world traffic from CAN, CAN-FD, and FlexRay can be regarded as future directions.

Chapter 5

A framework for integrating attacks and intrusion detection in CANoe

This chapter pursues the integration of adversary models and intrusion detection systems inside a CANoe simulation. The adversarial interventions are modelled using real-world CAN traffic that has been extracted from actual vehicles and then submitted to intrusion detection algorithms. The proposed IDS relies on the machine learning (ML) capabilities that are already supported by the MATLAB platform and is carried over to the CANoe simulation using C++ code. Such a unified platform that enables real-time simulation of both attacks and intrusion detection is advantageous since it offers a testbed for different intrusion detection algorithms. Given that CANoe is a standard tool in the automotive industry, the integration of these features sets room for realistic testing. The content of this chapter is based on the results published in a prior research paper by the author [19].

5.1 Data collection and their integration inside CANoe

In this section, the data collection procedure inside vehicles is presented. Then this section gives a brief overview of CAN network architecture and how the collected CAN traffic is used in CANoe simulation.

5.1.1 Data retrieval from the OBD2 port

The goal is to design and evaluate an IDS based on real-world CAN traces that would make the results more realistic in contrast to working with simulation based data. Therefore, the CAN bus data collection was performed through the OBD ports from several cars. The OBD port is dedicated to the collection of diagnostic information from all nodes and provides the status of several vehicle functions to the tester. As a result, there are many vehicles in which the OBD port has a direct connection to the main CAN bus.

To avoid security vulnerabilities that are exposed by the OBD port, a CAN network topology in which this port is linked to an ECU gateway responsible for the collection of diagnostic data from all other nodes, should be implemented. In this scenario, the OBD port would only expose diagnostic messages that are part of the UDS protocol and follow the request-response protocol. Many cars, however, lack such a gateway ECU in order to save additional costs, and the OBD port has access to the main CAN bus. The experiments carried inside cars in this chapter demonstrate that the OBD port provides access to the entire CAN communication (in the case of certain vehicles).

Before starting the data collection, the experiments were conducted to see whether the CAN communication is accessible through CAN pins of the OBD port and what the current bit rate is used. Using an oscilloscope, we observed that the CAN communication is exposed. Also, we noticed that one vehicle employs a baud rate of 250 Kbit/s, while the second operates at a baud rate of 500 Kbit/s. In the following phase, the CAN traffic was recorded for approximately 20 minutes while the cars were parked (stationary) and 20 minutes while they were moving. Many driver-specific activities, such as switching the car lights, accelerating and braking abruptly, gear shifts, etc., were carried out during this time. These were conducted in order to record more complex CAN traffic. The previous steps were performed on two types of vehicles: a sedan and an SUV.

Figure 5.1 shows the experimental setup. This accounts for a laptop that runs an application based on the Vector XL Driver Library, the Vector VN1630 USB-to-CAN interface, and the CAN cables which are connected with an OBD plug on one side and DB9 female connector on the other side. The DB9 female connector is compatible with the VN1630 device. The VN1630 device was designed by Vector, a reputable company that offers solutions for the development of automotive networks. This device belongs to the VN1600 family. Many software programs, including CANoe and CANape, as well as the XL Driver Library, provide support for the VN1630 device. The XL Driver Library enables the deployment of particular applications based on an Application Programming Interface (API) suitable for Vector's devices. Moreover, the library allows for interfacing with several protocols including CAN, CAN-FD, LIN, FlexRay, etc., and gives access to device functionalities such as message reception and transmission, various configuration parameters, e.g., bit rate, etc. Figure 5.2 depicts an example of messages that a simple application based on this library was able to capture through the OBD port from the vehicle. Each recorded CAN frame has the following structure:

- the CAN channel index where the frame was received,
- the timestamp having a resolution in nanoseconds,
- the frame identifier,
- the size of the datafield in bytes,
- the datafield carried by the frame.

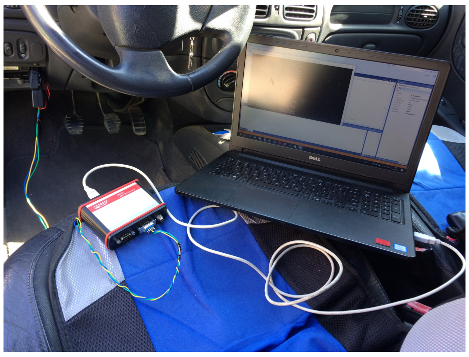
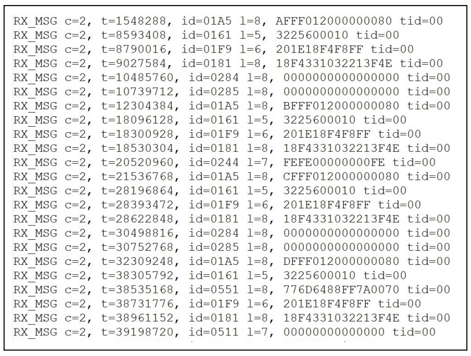


Figure 5.1: Data collection setup



```
RX_MSG c=2, t=1548288, id=01A5 l=8, AFFF012000000080 tid=00
RX_MSG c=2, t=8593408, id=0161 l=5, 3225600010 tid=00
RX_MSG c=2, t=8790016, id=01F9 l=6, 201E18F4F8FF tid=00
RX_MSG c=2, t=9027584, id=0181 l=8, 18F4331032213F4E tid=00
RX_MSG c=2, t=10485760, id=0284 l=8, 0000000000000000 tid=00
RX_MSG c=2, t=10739712, id=0285 l=8, 0000000000000000 tid=00
RX_MSG c=2, t=12304384, id=01A5 l=8, BFFF012000000080 tid=00
RX_MSG c=2, t=18096128, id=0161 l=5, 3225600010 tid=00
RX_MSG c=2, t=18300928, id=01F9 l=6, 201E18F4F8FF tid=00
RX_MSG c=2, t=18530304, id=0181 l=8, 18F4331032213F4E tid=00
RX_MSG c=2, t=20520960, id=0244 l=7, FEFE00000000FE tid=00
RX_MSG c=2, t=21536768, id=01A5 l=8, CFFF012000000080 tid=00
RX_MSG c=2, t=28196864, id=0161 l=5, 3225600010 tid=00
RX_MSG c=2, t=28393472, id=01F9 l=6, 201E18F4F8FF tid=00
RX_MSG c=2, t=28622848, id=0181 l=8, 18F4331032213F4E tid=00
RX_MSG c=2, t=30498816, id=0284 l=8, 0000000000000000 tid=00
RX_MSG c=2, t=30752768, id=0285 l=8, 0000000000000000 tid=00
RX_MSG c=2, t=32309248, id=01A5 l=8, DFFF012000000080 tid=00
RX_MSG c=2, t=38305792, id=0161 l=5, 3225600010 tid=00
RX_MSG c=2, t=38535168, id=0551 l=8, 776D6488FF7A0070 tid=00
RX_MSG c=2, t=38731776, id=01F9 l=6, 201E18F4F8FF tid=00
RX_MSG c=2, t=38961152, id=0181 l=8, 18F4331032213F4E tid=00
RX_MSG c=2, t=39198720, id=0511 l=7, 00000000000000 tid=00
```

Figure 5.2: Logged CAN frames

In order to prevent potential harm to the vehicle, the application was implemented solely for message reception while the transmission of CAN messages (injections) is disabled. Thus, the adversarial actions will be modelled using the CANoe simulation.

5.1.2 CAN network architecture and data usage in CANoe

The adversarial actions and intrusion detection features were embedded inside a CANoe-based simulation. The use of CANoe, i.e., an all-in-one integrated software solution, allows for the design, simulation, testing, and evaluation of in-vehicle networks. CANoe is the most common software solution used in the automotive industry by OEMs (original equipment manufacturers) to develop automotive networks.

CANoe makes use of the necessary building blocks to enable the modeling and real-time detection of real-world CAN bus attacks. Figure 5.3 shows the designed CAN network architecture for the analysis from this chapter. This network architecture is based on the following three blocks:

- the *replay node* – is implemented as a replay block, i.e. a specialized type of node which is responsible for replaying the recorded real-world CAN traffic,
- the *adversary model node* – is responsible for mimicking the real-world adversarial actions and is implemented using a CAPL (CAN Application Programming Language) node,
- the *IDS node* – is implemented utilizing a CAPL node and is responsible for evaluating and classifying the frames as either genuine or attack.

The replay block can be easily disabled for online attacks and traffic analysis when the CANoe simulated bus is connected to the in-vehicle bus through a VN device. CANoe-specific functionalities including events, system variables, message structures, and message databases are available by using CAPL, a C-based language [117].

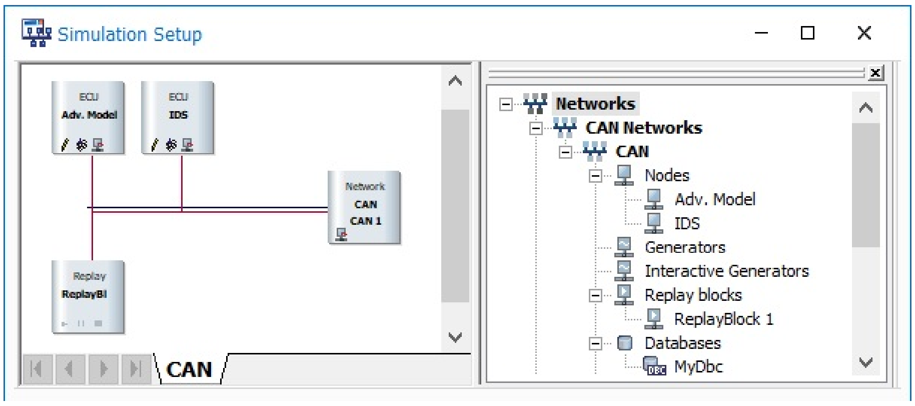


Figure 5.3: CAN network design inside CANoe simulation

5.2 Adversary Model

This section discusses the adversary model envisioned for the next experiments and provide an overview on how these adversarial bahaviors are integrated in CANoe.

5.2.1 Types of Attacks

Typically, adversary models rely on the Dolev-Yao model [118]. This model accounts for an adversary with complete network control. In other words, the adversary has the ability to intercept, block, replay, modify, or inject frames into the network. If any security measures exists, the adversary can only control them if he possesses the relevant keys. In the experiments from this chapter, we do not discuss any security mechanism because they are typically not present in the extracted CAN traffic from the vehicles and even if they were implemented, there would be no access to the manufacturer's specifications. Authentication mechanisms over the CAN bus, for instance, are typically regarded as confidential information.

The adversary envisioned for the next experiments has access to the CAN traffic recorded inside the cars and is developed in a similar vein to the current research on adversarial models for the CAN bus. These adversarial behaviors are integrated into the CANoe simulation. The evaluation takes into account the following types of attacks:

1. **Replay of regular CAN frames** occurs when the adversary intercepts the legitimates frames and then sends them on the CAN bus. The malicious frames in this instance are exact replicas of the legitimate ones sharing the same identifier and data field as legitimate frames. The periodicity of the CAN frames that have the same ID is the only sign that this kind of attack is taking place assuming that more messages with the same ID will be visible on the bus. The interface from the

CANoe simulation allows the configuration of the attacked frame identifier and the delay at which the malicious frame is injected after the reception of a legitimate one with the configured ID. The busload is increased as a result of the replay attacks, delaying or even stopping the transmission of other genuine frames.

2. **Injection attacks** are defined as insertions of adversarial messages on the CAN bus as we detail in the lines that follow:

   2.1. **Injection of random data**, also known in the literature as fuzzing attacks [116], is an attack in which the adversary intercepts legitimate frames and after that, injects the malicious frames on the CAN bus with a predetermined delay which can be configured from the interface. The identifier of the malicious frames is identical to that of legitimate frames, while the data field is randomly generated. The delay is represented by the amount of time between the genuine frame being intercepted and the attack frame being triggered for transmission. Furthermore, the delay at which the adversarial frame is transmitted can be configured with a resolution of 1 µs. Similar with the attack that was previously discussed, the ID of the targeted CAN frame can be selected using the graphical user interface (GUI).

   2.2. **Injection with scalar addition or multiplication of the data field content** - network nodes transmit information acquired from different sensors placed inside the vehicle, such as the speed sensor, fuel pressure, engine temperature, steering angle, and braking pressure, across the CAN bus. Due to the possibility of sensor signals to rely on a linear transfer function, the slope of such a function is a constant. As a result, in a real-world attack scenario, the payload bytes of the intrusion frame may be increased or multiplied by some constant values. Furthermore, delays can be configured to the adversarial messages.

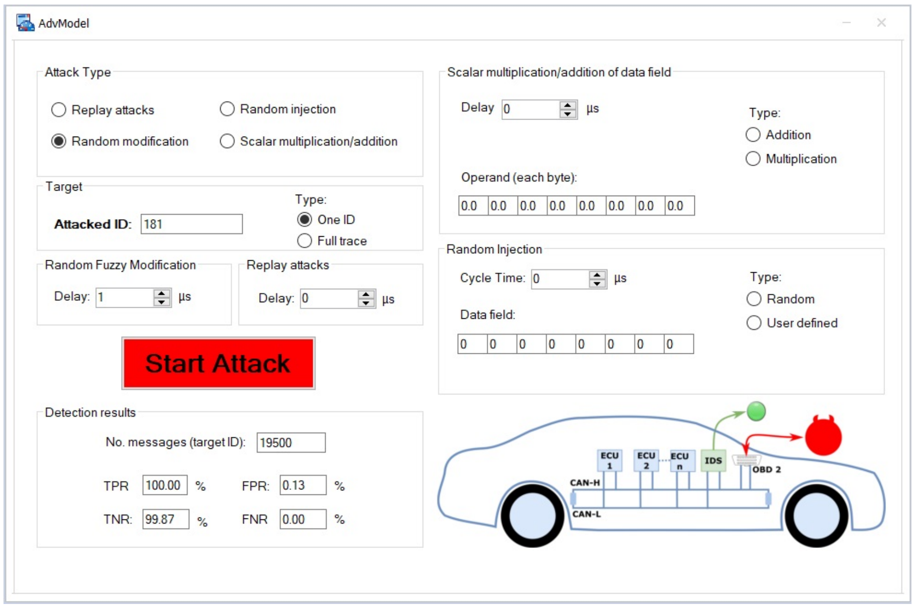
   2.3. **Arbitrary injection** - is the scenario in which the adversary can inject messages at discretion with a predefined or randomly generated payload and ID. Unlike the case of the aforementioned attacks, the transmission of the adversarial messages will occur periodically in accordance with the predefined cycle time.

Although not considered for the current evaluation, other attacks have been proposed in the literature as well. Now, we discuss why we did not take them into account, at least not at this time.

1. **DoS attacks** are easy to mount on the CAN bus. Since the principle of bus arbitration relies on CAN IDs to facilitate collision avoidance, continuously injecting messages with the highest priority ID (0x000), causes the bus to become unavailable and prevents regular frames from transmitting because of the loaded bus. Detecting an attack in which the ID 0x000 is injected, on the other hand, is straightforward. This

can be accomplished by simply looking for the successive transmissions of messages with this ID that does not appear in regular traces. Sending CAN frames with low priority IDs that are not null, nevertheless have a greater priority than genuine IDs, is a more refined variation of this attack. These attacks can be configured through the interface as arbitrary injections, which enables editing of both the ID and data field, but we do not regard them independently as DoS attacks (which may be a consequence). Although this kind of attacks is detectable by the IDS, the effect still persists because such attacks cannot be stopped due to the high priority IDs which will win the bus arbitration. Two recent works explore the use of relays to actively block intruders in performing such attacks [119], [120].

2. **Bus off attacks** are adversarial actions that result in legitimate nodes being placed in a bus-off state. This is possible because to the error management mechanism of CAN. The works [121], [122] demonstrate the feasibility of such attacks. Although examining such adversarial actions could be interesting, we lack information about the behavior of each ECU and more importantly, of the car, in such situations (this can be tested only in the actual car and known by the manufacturer). The network that we use is simulated only based on the collected traces that do not exhibit this behavior. Furthermore, the only way to defend against such attacks is to change the CAN error-handling system, which is outside the scope of the current chapter.


AdvModel

Attack Type
Replay attacks [✓ Random modification] Random injection Scalar multiplication/addition

Target
Attacked ID: 181
Type: [✓ One ID] Full trace

Random Fuzzy Modification
Delay: 1 µs

Replay attacks
Delay: 0 µs

Start Attack

Detection results
No. messages (target ID): 19500
TPR 100.00 % FPR: 0.13 %
TNR: 99.87 % FNR 0.00 %

Scalar multiplication/addition of data field
Delay 0 µs
Type: Addition Multiplication
Operand (each byte): 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0

Random Injection
Cycle Time: 0 µs
Type: Random User defined
Data field: 0 0 0 0 0 0 0 0

Figure 5.4: Graphical user interface for the CANoe simulation

5.2.2 Application Interface

Figure 5.4 depicts the application interface deployed inside the CANoe simulation to enable the configuration of the adversary and IDS nodes. To create an user-friendly interface, we use standard graphic controls, such as radio buttons and combo boxes. System variables, which may be accessed by particular CAPL functions and events, are used to facilitate the connection between the graphical user interface and the CAPL code. As a result, the proposed adversary model offers a variety of attacks and is capable of several actions like reading, altering, and replaying CAN frames. The user must first choose the attack type that will be mounted. Beside this, the user can configure whether the attack should be performed on a given ID or on all frames from the trace. Certain parameters can be adjusted for every attack type.

The IDS node, on the other hand, is responsible for classifying each message as an attack or a legitimate one, during the simulation run. The alarm led will be either red or green depending on the classification of the received frame as either attack or genuine. Also, the detection rates and the total number of classified messages are displayed when the simulation has ended.

5.3 Intrusion Detection Algorithms: Background and Tools

In this section, the tools used in the current evaluation as well as the MATLAB-CANoe integration are presented. Additionally, an overview of the k-NN algorithm as a candidate for intrusion detection algorithm is discussed.

5.3.1 Statistics and Machine Learning Toolbox

The *Statistics and Machine Learning Toolbox* from MATLAB was used to implement the intrusion detection system. This toolbox offers a variety of ML based approaches. These ML algorithms can be supervised or unsupervised learning-based, and we decide to use k-NN for this evaluation. The decision to use k-NN was made since it is typically employed when the input data is unknown. Indeed, in this instance, the information is derived from the CAN traffic that was recorded inside the cars and we lack access to the manufacturer's specifications. As a result, nothing about the data has been known in advance. The CAN frames injected by the adversary CAPL node were marked in order to generate the training trace.

In the training stage, the supervised learning (used by the k-NN based approach as well) uses a set of n pairs as its observation samples. Mathematically, a pair can be described as follows: $\{(i_0, o_0), (i_1, o_1), ...(i_{n-1}, o_{n-1})\}$, where each pair contains the input and the expected output. The result of the training stage is a model (a trained function), which is able of making predictions over fresh data that will serve as inputs during the test stage.

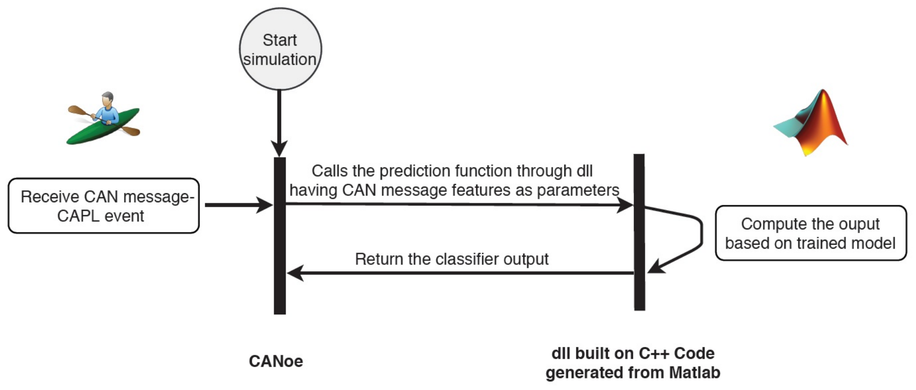

Start simulation
Receive CAN message-CAPL event
Calls the prediction function through dll having CAN message features as parameters
Return the classifier output
Compute the ouput based on trained model
CANoe
dll built on C++ Code generated from Matlab

Figure 5.5: An illustration of the data flow between MATLAB and CANoe using dll

In the evaluation from this chapter, we also account for the MATLAB's ability to generate C/C++ code. Consequently, the C/C++ code implements the trained model and based on the generated code, a dynamic library *dll* is created. Following that, the *dll* is integrated into the CANoe simulation with the help of CAPL code.

Integrating a customized library into CANoe has the benefit of allowing access to system resources such as CPU and memory [123], that are not readily available in CANoe. The data flow exchanged between CANoe and the MATLAB-based library in order to embed the prediction function with the features extracted from the received CAN frames, is shown in Figure 5.5.

5.3.2 k-NN Algorithm

The evaluation of the proposed IDS relies on the k-NN algorithm. This algorithm is frequently used in classification problems and sometimes even in network IDS [124]. Basically, the k-NN algorithm employs a metric for measuring distance, such as the Euclidean, Hamming, Minkowski, Jaccard, etc. We note that the Euclidean distance is used in the majority of the experiments that follow, although changing to any of the aforementioned distances is easy to do.

Typically, a ML based approach has two phases: the training phase and the testing phase. As a result, the entire CAN trace was divided into training and testing parts. For the experiments that follow, the first phase is carried out offline to facilitate the training of the k-NN algorithm using inputs-output samples. In this phase, each input is assigned to the true class c (legitimate or attack message). As a result, this phase ends with the generation of the k-NN model. Real-time detection using the generated model represents the second phase. At this point, the decision rule from the model is used to map each input to the predicted class $\hat{c}$. According to how many neighbors k there are, the decision rule is as follows:

- Decision rule when *k=1*: considering that m_t is a test message and m_i is a training message, then m_n is nearest neighbor to m_t if and only if the following condition related to the Euclidean distance is satisfied:

$$d_e(m_t, m_n) = min_i\{d_e(m_t, m_i)\}, \tag{5.1}$$

  where i represent the index of all training frames. The predicted class $\hat{c}$ for the m_t based on the trained model is equal with the true class c of the m_i with the smallest Euclidean distance to m_t.

- Decision rule when *k>1*: the predicted class $\hat{c}$ for the m_t based on the trained model is equal with the most frequently c among the *k* nearest training frames.

The k-NN input sample is an array that takes into account the payload as well as the time interval between successive timestamps of frames with the same ID. In this instance, the mathematical description of the input is as follows:

$$I = \{i_0, i_1, i_2, ..., i_8\}, \tag{5.2}$$

where i_0 stands for the time interval and $i_1...i_8$ stands for every byte from the payload. To facilitate a clear majority, the evaluation was carried out using an odd number of neighbors. (for example 1, 3, 15).

5.4 Experimental results

This section presents the experiments that are conducted to test the proposed IDS and the detection results, respectively.

5.4.1 Results on detecting intrusions

The experiments were devised to evaluate all of the adversary models that had been previously described. There are multiple scenarios for each type of attack. These depend on the delay of the injected message. Also, multiplication or addition coefficients can be applied on the bytes from the payload. Due to the lack of extended frames in the recorded traces from the cars, only standard frames were used for this evaluation. The datasets were generated to cover the scenarios in which the adversary targets either a particular CAN ID or the full trace, that is, all CAN IDs. This was done by injecting attack frames using the CANoe simulation. The results on detecting intrusions when a single CAN ID is targeted, relies on parts of traces that are split as follows: 500 frames for training phase and 19500 frames for testing phase. The rationale behind selecting a small percentage for

Table 5.1: Results on detecting intrusions for the defined types of attacks

	Attack params.				k-NN Parameters		Detection rates			
No.	One/Full ID	Att. type	Operand	Delay (ms)	No. neigh.	Distance	TNR (%)	TPR (%)	FPR (%)	FNR (%)
1.	one	r	n/a	9.750	1	Euclidean	99.00	99.65	1.00	0.35
2.	one	r	n/a	0.001	1	Euclidean	100	100	0	0
3.	one	r	n/a	0.001	1	Euclidean	88.86	100	11.14	0
4.	one	r	n/a	5	1	Euclidean	88.88	100	11.12	0
5.	one	r	n/a	9	1	Euclidean	90.33	83.47	9.67	16.53
6.	one	r	n/a	9.750	1	Euclidean	87.98	51.88	12.02	48.12
7.	one	r	n/a	50	1	Euclidean	88.31	84.66	11.69	15.34
8.	one	ir	n/a	0.001	1	Euclidean	99.87	100	0.13	0
9.	one	ir	n/a	9.750	1	Euclidean	99.87	100	0.13	0
10.	one	isa	$\alpha = 2$	0.001	1	Euclidean	89.63	100	10.37	0
11.	one	isa	$\alpha = 2$	9.750	1	Euclidean	91.34	53.98	8.66	46.02
12.	one	ism	$\alpha = 2$	0.001	1	Euclidean	89.65	100	10.35	0
13.	one	ism	$\alpha = 2$	9.750	1	Euclidean	91.38	67.74	8.62	32.26
14.	one	isa	$\alpha = 2$	9.750	1	E(Δt), H(data)	90.72	100	9.28	0
15.	one	ism	$\alpha = 2$	9.750	1	E(Δt), H(data)	90.75	85.87	9.25	14.13
16.	full	r	n/a	0.001	1	Euclidean	95.21	100	4.79	0
17.	full	r	n/a	5	1	Euclidean	95.45	100	4.55	0
18.	full	r	n/a	9.750	1	Euclidean	95.23	66.58	4.77	33.42
19.	full	r	n/a	$\{$9.75, 19.75, 39.75, 99.75$\}$	1	Euclidean	94.76	50.06	5.24	49.94
20.	full	ir	n/a	0.001	1	Euclidean	99.53	100	0.47	0
21.	full	ir	n/a	5	1	Euclidean	99.40	100	0.6	0
22.	full	ir	n/a	9.750	1	Euclidean	99.57	91.52	0.43	8.48

training data is to address the more realistic case in which the IDS is trained for a brief amount of time, such as during manufacturing, and then operates for the entire life of the vehicle. The experiments performed on a single CAN ID aim for the attacks that are only mounted on CAN frames with a periodicity of 10 ms because this is a typical cycle time used in vehicular applications. Nevertheless, similar results will probably be produced for other cycles. In case of the full trace analysis, the experiments are based on 500 frames for training as well as 45000 frames for testing.

In what follows, the results obtained for replay injections are discussed. These results are shown in Table 5.1 while the extended results are presented in Table 5.2. In this scenario, the training stage was carried out on traces that have both legitimate and replay attack frames that were transmitted 9.750 ms and 0.001 ms after the genuine frame, respectively. Specifically, the results for the 9.750 ms delay are presented in row 1 from Table 5.1 and rows 1-2 from Table 5.2 while for 0.001 ms delay the results are shown in row 2 from Table 5.1 and rows 3-4 from Table 5.2. The first delay was precisely envisioned so that the frame manipulated by adversary would arrive on the CAN bus just before the legitimate frame, which is scheduled to arrive regularly at a cycle time of 10 ms and typically $\approx$ 250 μs is the time spent by the frame on the physical bus. The goal for the second one, i.e, 0.001 ms, is to make sure that the attack frame comes after the real frame right away. As inputs for the training stage, we use the payload content as well as the interval (Δt) between the timestamps of the successive frames defined by the targeted

CAN ID. The true positive rate is around 99% for the attacks with a delay of 9.750 ms and 100% for attacks with 0.001 ms delay. In the first scenario there is a false positive rate of about 1%, while for the second scenario there are no false positives. For the attacks with a 9.750 ms delay, there is a small amount of false positives because the frames manipulated by the adversary are injected roughly at the same time as the legitimate ones. As a result, sometimes the genuine frame and the malicious frame are mismatched. The excellent detection performance is due to the injection of malicious frames with a fixed latency, which is the same that was applied in the training stage as well (which is arguably too simple and unrealistic).

As a result, the following step in the IDS assessment was conducted to examine a more realistic scenario. So, the classifier was trained using a delay of 9.750 ms, whereas the testing frames were generated using a different delay of 9 ms. As anticipated, the true positive rates decreases significantly, i.e, below 20%. Considering that the IDS must enable the detection of attacks frames transmitted with any latency, the solution to this challenge was to train the classifier using traces generated with replay injections that take place at a random delay, covering the entire range between 0 to the cycle time of the message. All the experiments that follow are conducted using the traces produced with randomized delays. Table 5.1 (rows 3-7) contains the results for the given scenario, while Table 5.2 (rows 5–14) contains the extended results. The detection rate is close to 100% and the false positive rate is roughly 10% for delays of 0.001 ms and 5 ms, respectively. This percentage of false positives is brought on by the same payload of both genuine and attack frames.

For the next steps in the evaluation of the IDS, the adversarial interventions are more meticulously planned. The adversarial actions are refined so that injections takes place just before the genuine frame, i.e, 9 ms delay, or even in some circumstances sufficiently near to overlap with the legitimate message , i.e., 9.750 ms delay. The results for this scenario are presented in rows 5-6 from Table 5.1 as well as in rows 9-12 from Table 5.2. The performance of the IDS goes down at a level in which the detection rate decreases under 80% for the first scenario and to approximately 50% for the second one. Most of the results also show that, along with an increase in the amount of neighbors, there is a minor decrease in FPR and sometimes a more noticeable decrease in TPR. A suggestive example can be seen by comparing the row 5 from Table 5.1 where the detection rate is 83% and row 9 from Table 5.2 where the TPR drops to 52%.

For the fuzzing attacks the results are presented in rows 8-9 from Table 5.1. The extended results for this type of attack are shown in rows 15-18 from Table 5.2. In this instance, the results exhibit a true positive rate of roughly 100% for both investigated delay conditions. Furthermore, there are very few false positives in the results. The high randomness present inside the payload of attack frames which differs from the legitimate messages, justifies the high performance in detecting intrusions. Typically, compared to replay attacks, the fuzzing attacks are considerably trivial to be detected.

Since the experiments were performed using a relatively small value for the scalar (and hence only minor modifications are produced in the bytes of the datafield), the results for injections with scalar addition or multiplication of the payload show lower performance in detecting intrusions, as anticipated. The results for these scenarios are presented in rows 10-15 from Table 5.1 and the extension of the results can be shown in rows 19-30 from Table 5.2. On initial inspection, the results are very comparable to those obtained for the case of replay injections using the same delays: 0.001 ms and 9.750 ms. This may be accounted for by the fact that the data field has a lower impact on the output from the prediction function than the message frequency. This occurs because adding $\alpha = 2$ to every byte from the payload does not significantly change the Euclidean distance. Since a bigger α will inevitably result in significant changes of the datafield and then detecting the attacks will be straightforward, the experiments are conducted using a small value, i.e., $\alpha = 2$, to ensure that the payload of the message would only be slightly altered. The operation of scalar multiplication with $\alpha = 2$ has a bigger effect on the result of applying the Euclidean distance. As a result, for injections with scalar multiplication of datafield, as can be seen in row 13 from the Table 5.1, the detection rate is improved, i.e, around 67%.

An improvement in the performance of detecting such intrusions can be accomplished by using two trained models. The first model is based only on Δt utilizing Euclidean distance while the second focuses on the data field and use the Hamming distance. Each model classifier in the given scenario predicts a class for every message. The final predicted class based on aforementioned models is:

$$\hat{c} \in \hat{c}_1 \vee \hat{c}_2, \tag{5.3}$$

where $\hat{c}_1$ represents the predicted class for the first model and $\hat{c}_2$ corresponds to the second model.

With the help of this strategy, the sensitivity is increased to 100% for scalar addition and to 85% for multiplication as it is shown in rows 14–15 of Table5.1. For this scenario the amount of false positives is still at around 10%. We represent the Euclidean distance on the time interval between consecutive frames having same CAN ID with E(Δt) and the Hamming distance on data field with H(data), respectively.

The attack detection on a single CAN ID would require one trained model for each CAN ID and result in the need for significant resources in terms of computation and memory, which may not be available in resource constraint devices, i.e., automotive ECUs. As a result, the experiments are conducted to test the detection performance over the full trace, i.e, all CAN IDs.

The recorded trace includes frames with cycle times of 10, 20, 40, and 100 ms. Next, we discuss how the complete attack trace was generated. In the CAPL code, the attack probability is declared as a constant $P_r(A)$ for each received CAN frame. Then, a variable $\epsilon \in \big[0, 100\big]$, is initialized with random generated values. For every received CAN frame, the attack is triggered if the value of ϵ is less than or equal to the value of $P_r(A)$, else, no attack is raised. For this evaluation, the defined attack probability is $P_r(A) = 30$. As a result of testing the full trace, before the Δt and data field, the CAN ID is considered in the input of the classifier.

For replay attack over the full trace, the results are shown in rows 16-19 from Table 5.1 as well as rows 31-38 from Table 5.2. On the other hand, for the case of fuzzing attacks, the results are presented in rows 20-22 from Table 5.1 and rows 39-44 from Table 5.2. Even if the evaluation was performed over the full trace, using a single trained model, the results are still acceptable. For the scenarios in which 0.001 ms and 5 ms delays were used for replay attacks, the detection performance is comparable to the one obtained on replays over a single ID, or even superior when a 9.750 ms delay was used. Indeed, this is the case, because the injected frame, which has a 9.750 ms delay, reaches the bus before the regular frame or even overlaps with it exclusively in the case of frames that have a 10 ms cycle, whereas the entire trace includes frames with different cycle times values (20,40, 100 ms), for which the injected frame is much more obvious. Of course, the adversarial actions can be clever, such that the delays of the injected frames are more roughly comparable to the cycle time of each frame. The results obtained for this scenario are shown in the row 18 from Table 5.1 as well as rows 37-38 from Table 5.2. In the case of replay injection, the results exhibit a lower detection, roughly 50%, which is approximately 2% less than the case of a single monitored ID. With the exception of a 9.750 ms delay, where it decreases to around 90% when one neighbor is employed and to 60% when multiple neighbors are employed, the detection rates for fuzzing attacks are generally roughly 100%.

5.5 Concluding remarks

The proposed framework, which combines adversarial models with IDS inside a CANoe simulation, proves to be useful for realistic testing. This is also strengthened by the fact that the attacks traces were based on real-world CAN traffic. Another benefit of this framework is the avoidance of any risk for harming the vehicles and passengers. Therefore, evaluating adversarial behaviors in a simulation setup is safer and easier.

Allocating particular portions of CAN traffic to specific ECUs is interesting for future work because it would open the road for targeted attacks against particular ECUs. A comprehensive simulation that accounts for the behavior of each ECU is a more difficult goal, but seems to be feasible in the future. Using ML functionalities, made available by MATLAB, for the classification of CAN frames is an effective approach for implementing and testing such intrusion detection algorithms because of the extensive ML toolkit that MATLAB provides. Moreover, a future direction for extending the proposed framework, is to add other intrusion detection algorithms.

Table 5.2: Results on detecting intrusions for the defined types of attacks (k-NN with 3 or 15 neighbors)

	Attack params.			No. messages		k-NN Parameters		Detection rates			
No.	Att. type	Operand	Delay ms	training	testing	No. neigh.	Distance	TNR	TPR	FPR	FNR
1.	r	n/a	9.750	500	19500	3	Euclidean	98.55%	99.31%	1.45%	0.69%
2.	r	n/a	9.750	500	19500	15	Euclidean	97.55%	97.83%	2.45%	2.17%
3.	r	n/a	0.001	500	19500	3	Euclidean	100%	100%	0%	0%
4.	r	n/a	0.001	500	19500	15	Euclidean	100%	100%	0%	0%
5.	r	n/a	0.001	500	19500	3	Euclidean	90.02%	100%	9.98%	0%
6.	r	n/a	0.001	500	19500	15	Euclidean	91.32%	100%	8.68%	0%
7.	r	n/a	5	500	19500	3	Euclidean	89.99%	100%	10.01%	0%
8.	r	n/a	5	500	19500	15	Euclidean	91.30%	100%	8.70%	0%
9.	r	n/a	9	500	19500	3	Euclidean	91.61%	52.53%	8.39%	47.47%
10.	r	n/a	9	500	19500	15	Euclidean	91.33%	31.11%	8.67%	68.89%
11.	r	n/a	9.750	500	19500	3	Euclidean	89.33%	50.67%	10.67%	49.33%
12.	r	n/a	9.750	500	19500	15	Euclidean	91.26%	50.67%	8.74%	49.33%
13.	r	n/a	50	500	19500	3	Euclidean	89.96%	83.75%	10.04%	16.25%
14.	r	n/a	50	500	19500	15	Euclidean	91.40%	83.75%	8.60%	16.25%
15.	ir	n/a	0.001	500	19500	3	Euclidean	99.70%	100%	0.30%	0%
16.	ir	n/a	0.001	500	19500	15	Euclidean	98.63%	100%	1.37%	0%
17.	ir	n/a	9.750	500	19500	3	Euclidean	99.70%	100%	0.30%	0%
18.	ir	n/a	9.750	500	19500	15	Euclidean	98.63%	100%	1.37%	0%
19.	isa	$\alpha = 2$	0.001	500	19500	3	Euclidean	91.37%	100%	8.63%	0%
20.	isa	$\alpha = 2$	0.001	500	19500	15	Euclidean	91.13%	100%	8.87%	0%
21.	isa	$\alpha = 2$	9.750	500	19500	3	Euclidean	92.15%	51.01%	7.85%	48.99%
22.	isa	$\alpha = 2$	9.750	500	19500	15	Euclidean	91.09%	50.40%	8.91%	46.60%
23.	ism	$\alpha = 2$	0.001	500	19500	3	Euclidean	91.39%	100%	8.61%	0%
24.	ism	$\alpha = 2$	0.001	500	19500	15	Euclidean	91.12%	100%	8.88%	0%
25.	ism	$\alpha = 2$	9.750	500	19500	3	Euclidean	92.16%	60.70%	7.84%	39.30%
26.	ism	$\alpha = 2$	9.750	500	19500	15	Euclidean	91.11%	50.59%	8.89%	49.41%
27.	isa	$\alpha = 2$	9.750	500	19500	3	E(Δt), H(data)	91.07%	100%	8.93%	0%
28.	isa	$\alpha = 2$	9.750	500	19500	15	E(Δt), H(data)	91.06%	100%	8.94%	0%
29.	ism	$\alpha = 2$	9.750	500	19500	3	E(Δt), H(data)	91.06%	85.52%	8.94%	14.48%
30.	ism	$\alpha = 2$	9.750	500	19500	15	E(Δt), H(data)	91.09%	85.17%	8.91%	14.83%
31.	r	n/a	0.001	5000	45000	3	Euclidean	96.74%	100%	3.26%	0%
32.	r	n/a	0.001	5000	45000	15	Euclidean	98.77%	100%	1.23%	0%
33.	r	n/a	5	5000	45000	3	Euclidean	96.75%	100%	3.25%	0%
34.	r	n/a	5	5000	45000	15	Euclidean	98.70%	100%	1.30%	0%
35.	r	n/a	9.750	5000	45000	3	Euclidean	96.79%	65.33%	3.21%	34.67%
36.	r	n/a	9.750	5000	45000	15	Euclidean	98.83%	35.95%	1.17%	64.05%
37.	r	n/a	$\{$9.75, 19.75, 39.75, 99.75$\}$	5000	45000	3	Euclidean	96.40%	48.94%	3.60%	51.06%
38.	r	n/a	$\{$9.75, 19.75, 39.75, 99.75$\}$	5000	45000	15	Euclidean	98.68%	47.41%	1.32%	52.59%
39.	ir	n/a	0.001	5000	45000	3	Euclidean	99.47%	100%	0.53%	0%
40.	ir	n/a	0.001	5000	45000	15	Euclidean	99.57%	100%	0.43%	0%
41.	ir	n/a	5	5000	45000	3	Euclidean	99.31%	100%	0.69%	0%
42.	ir	n/a	5	5000	45000	15	Euclidean	99.49%	100%	0.51%	0%
43.	ir	n/a	9.750	5000	45000	3	Euclidean	99.53%	86.46%	0.47%	13.54%
44.	ir	n/a	9.750	5000	45000	15	Euclidean	99.63%	60.97%	0.37%	39.03%

Chapter 6

Intrusion detection and prevention on SAE J1939 CAN buses

This chapter presents a solution for securing J1939 CAN buses and is based on the results of the author published in [18]. The solution relies on a two-stage IDS that is complemented by an active prevention mechanism that eliminates the intrusions in real-time. In order to get realistic detection results, the proposed solution is evaluated based on real-world J1939 CAN traffic that was collected inside an agricultural heavy-duty vehicle. Computational results on four automotive-graded development boards for evaluating the applicability of the proposed solution are provided in this chapter. A proposal for an algorithm that relies on an input capture unit (ICU) and interprets the content of the CAN frames before the receivers place the dominant bit in the ACK slot, so that an intrusion frame can be effectively destroyed before reception, is also discussed. Therefore, this algorithm sets room for the real-time intrusion prevention. The designed prototype is tested on attacks that are mounted in a laboratory setup.

6.1 Data collection and analysis

This section describes the data collection procedure and after that, it presents an investigation of J1939 specific features and a quantitative analysis, which are performed over the collected CAN traffic.

6.1.1 Data extraction from the J1939 specific diagnostic port

The goal is to design an intrusion detection and prevention system (IDPS) tailored to J1939 specifications and evaluate it based on real-world CAN traces. This approach would provide more realistic results in contrast to relying solely on simulation data. For a real-world instantiation of J1939 CAN bus traffic, the data collection was performed through the specific OBD port from a recent agricultural vehicle produced by a reputable manufacturer. Although we avoid to provide the identify of the manufacturer due to the confidentiality concerns, the recorded CAN traffic complies with J1939 requirements.

Prior to initiating data collection, a fast verification using an oscilloscope was performed to check if the CAN traffic is accessible through the 9-pin diagnostic port, which is common in J1939 based deployments [111]. Figure 6.1 shows the pinout diagram of the diagnostic port connector in accordance with J1939 standards, highlighting the pins used for CAN communication. The oscilloscope based examination was conducted to determine the availability of CAN traffic on each CAN channel, i.e, pairs C-D and H-J. The evaluation shows that only the main channel (pin pair C-D), which operates at a bit rate of 250 Kbps, exposes the CAN communication. Regarding the second CAN channel (pin pair H-J) that is allocated for OEM (original equipment manufacturer) specific solutions, no CAN communication was detected.

After connecting to the J1939 diagnostic port, the CAN traffic was recorded for roughly 30 minutes. Various generic driving-related and tractor-specific tasks, including managing the bucket and moving forward or backward, were carried out throughout this time interval. The previously described actions were conducted in an effort to activate as many events as possible that will lead to a more complex CAN traffic.

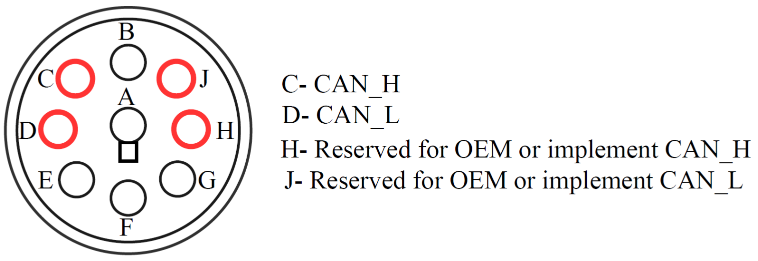

B
C
J
A
D
H
E
G
F
C- CAN_H
D- CAN_L
H- Reserved for OEM or implement CAN_H
J- Reserved for OEM or implement CAN_L

Figure 6.1: J1939 9-pin diagnostic port

The data collection setup is shown in Figure 6.2 and includes a Vector VN1640 USB-to-CAN interface, a laptop that runs the logging software application implemented based on the Vector XL Driver Library as well as the CAN connection (cables) between the VN1640 and the J1939 diagnostic port.

6.1.2 Examination of J1939 specific features in collected traffic

The recorded CAN data includes 51 CAN IDs, out of which 41 appear at runtime, while the remaining 10 IDs are only visible at startup. Subsequently, an interpretation of the these CAN IDs in accordance with the J1939 ID breakdown, is outlined. The ID breakdown is shown in Table 6.1. In order to identify the ECUs that are connected in the CAN network and are accessible from the outside world via the diagnostic port, the unique

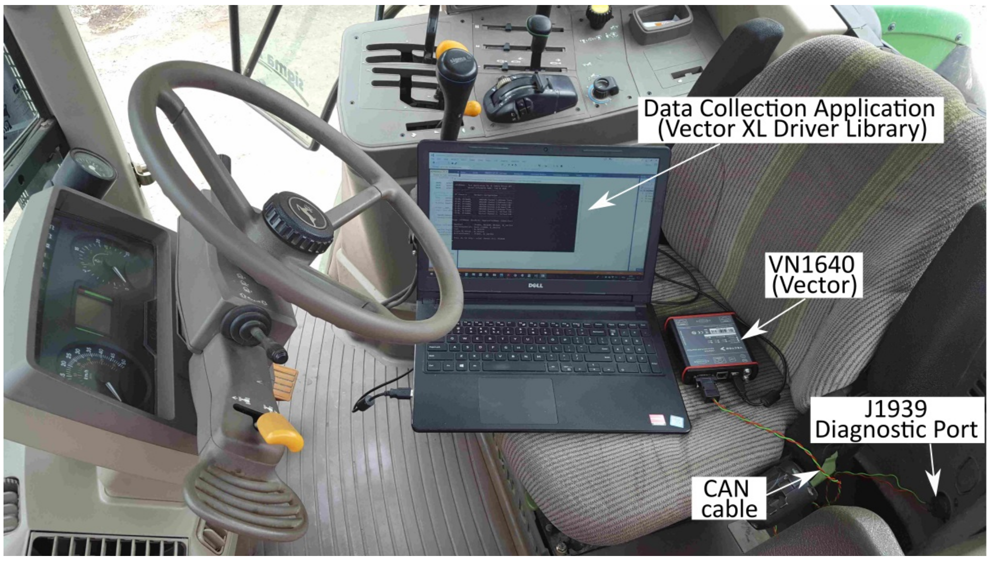


Figure 6.2: Experimental setup used for data collection

source addresses (SA) are examined according to the CAN IDs analysis. As a result, the recorded CAN communication features three distinct source addresses: 0x00, 0x03, and 0x21. These details related to source addresses are provided in the J1939 Digital Annex document [113]. Consequently, in accordance with [113], the following tasks are assigned to each ECU that is connected to the CAN network:

1. Engine Control Module (ECM) – source address 0x00,
2. Body Control Module (BCM) – source address 0x21,
3. Transmission Control Module (TCM) – source address 0x03.

In what follows, a brief description of the detected ECUs is provided:

1. *ECM* – is one of the most important ECU inside cars. It collects data from various engine sensors and evaluates them to adjust the engine actuators through the air-fuel ratio, fuel injection, ignition timing and variable valve timing parameters in order to achieve the best engine performance.
2. *BCM* – is the main ECU responsible for controlling a number of distinct body related operations, including interior and exterior illumination, power windows, seat position, climate control as well as central locking.
3. *TCM* – controls the automatic transmission and receives data from several sensors including wheel speed, throttle position, turbine speed and others as inputs. These

inputs are used by the TCU to produce an optimal performance by switching the gears. This is accomplished by controlling different outputs, including torque converter, clutch solenoid, shift lock, etc.

Table 6.1: ID breakdown according to [113] of the identified J1939 CAN packets

No.	Pr.	ID	PF	PS	PDU1	PDU2	DA	GE	SA	TP PGN	PG description
1.	6	0x18EEFF03	238	255	✓	–	GLB	–	TCM	–	Address Claimed
2.	6	0x18EEFF00	238	255	✓	–	GLB	–	ECM	–	Address Claimed
3.	6	0x18EEFF21	238	255	✓	–	GLB	–	BCM	–	Address Claimed
4.	6	0x18EAFF00	234	255	✓	–	GLB	–	ECM	–	PGN Request
5.	6	0x18EAFF03	234	255	✓	–	GLB	–	TCM	–	PGN Request
6.	6	0x18FE0F21	254	15	–	✓	–	15	BCM	–	Language Command
7.	6	0x18FECA03	254	202	–	✓	–	202	TCM	–	Active Diagnostic Trouble Codes (DM1)
8.	6	0x18FECA21	254	202	–	✓	–	202	BCM	–	Active Diagnostic Trouble Codes (DM1)
9.	6	0x18FEF200	254	242	–	✓	–	242	ECM	–	Fuel Economy (Liquid)
10.	3	0xCF00400	240	4	–	✓	–	4	ECM	–	Electronic Engine Controller 1
11.	6	0x18FEF121	254	241	–	✓	–	241	BCM	–	Cruise Control/Vehicle Speed
12.	7	0x1CFEC303	254	195	–	✓	–	195	TCM	–	Electronic Transmission Controller 5
13.	3	0xCF00300	240	3	–	✓	–	3	ECM	–	Electronic Engine Controller 2
14.	6	0x18F00503	240	5	–	✓	–	5	TCM	–	Electronic Transmission Controller 2
15.	6	0x18FEDF00	254	223	–	✓	–	223	ECM	–	Electronic Engine Controller 3
16.	3	0xCFE4521	254	69	–	✓	–	69	BCM	–	Primary or Rear Hitch Status
17.	6	0x18FEF021	254	240	–	✓	–	240	BCM	–	Power Takeoff Information 1
18.	6	0x18FEF021	254	239	–	✓	–	239	ECM	–	Engine Fluid Level/Pressure 1
19.	3	0xCFE4421	254	68	–	✓	–	68	BCM	–	Front Power Take off Output Shaft
20.	3	0xCFE4321	254	67	–	✓	–	67	BCM	–	Rear Power Take off Output Shaft
21.	6	0x18FEF600	254	246	–	✓	–	246	ECM	–	Intake/Exhaust Conditions 1
22.	6	0x18FEAE21	254	174	–	✓	–	174	BCM	–	Air Supply Pressure
23.	7	0x1CFDDF21	253	223	–	✓	–	223	BCM	–	Front Wheel Drive Status
24.	6	0x18FEFC21	254	252	–	✓	–	252	BCM	–	Dash Display 1
25.	6	0x18FEF721	254	247	–	✓	–	247	BCM	–	Vehicle Electrical Power 1
26.	6	0x18F00621	240	6	–	✓	–	6	BCM	–	Electronic Axle Controller 1
27.	3	0xCFDCC21	253	204	–	✓	–	204	BCM	–	Operators External Light Controls
28.	6	0x18EAFF21	234	255	✓	–	GLB	–	BCM	–	PGN Request
29.	6	0x18FEE500	254	229	–	✓	–	229	ECM	–	Engine Hours, Revolutions
30.	6	0x18FEEE00	254	238	–	✓	–	238	ECM	–	Engine Temperature 1
31.	6	0x18FEF700	254	247	–	✓	–	247	ECM	–	Vehicle Electrical Power 1
32.	7	0x1CECFF00	236	255	✓	–	GLB	–	ECM	CM.BAM	Engine Configuration 1, VIN
33.	7	0x1CEBFF00	235	255	✓	–	GLB	–	ECM	TP.DT	Engine Configuration 1, VIN
34.	7	0x1CECFF21	236	255	✓	–	GLB	–	BCM	CM.BAM	Vehicle Identification Number
35.	7	0x1CEBFF21	235	255	✓	–	GLB	–	BCM	TP.DT	Vehicle Identification Number

Additionally, the collected CAN data contains J1939 specific multi-frame messages, and the analysis shows that, as anticipated, the Broadcast Data Transfer was being used as a transport protocol. Next, the analysis goes through a few specifics regarding the recorded IDs. The examination of IDs shows that there are relevant messages for the address claiming procedure (rows 1-3) from Table 6.1 as well as the corresponding PGN request (rows 4-5 and 28) from the same table. There are numerous other IDs that occur after the address claiming process and provide data about the engine or transmission, air supply, cruise control, external light controls and many others. The Engine Configuration 1 and Vehicle Identification Number (VIN) PGNs, which are shown in rows 32 and 33 of Table 6.1, are transmitted by the ECM through multi-frame based protocols, which are J1939 specific as well. The BCM also sends the VIN as can be seen in rows 34-35 from Table 6.1. Also, the collected traffic includes two diagnostic related J1939 frames that carry the active diagnostic trouble codes transmitted by the BCM and ECM. These are shown in rows 7-8 from Table 6.1 and are part of the on-board diagnostic subsystem. The 16 OEM specific IDs are absent from 6.1 since the standard does not define their function.

6.1.3 A quantitative analysis of collected traffic

The prior examination of the collected CAN data was conducted in accordance with the J1939 standard collection and describes aspects at a logical level that are useful in traffic reconstruction. In what follows, a quantitative analysis of the captured traffic is done. This enables to impose certain restrictions in the design of the intrusion detection and prevention system.

Typically, in-vehicle ECUs transmit CAN messages with a predefined periodicity for each particular CAN ID. This principle also extends to the J1939 deployment under investigation in this study. Following a comprehensive examination of the traffic, CAN IDs with the cycle times of 20, 25, 50, 100 or 500 ms were identified. Additionally, a small part of the IDs exhibits a greater cycle time, being broadcast at 1 second time intervals. Rarely, multi-frame IDs are transmitted at a cycle of 5 s, and, as anticipated, a burst of frames follows after the BAM message. The plots displayed in Figure 6.3 demonstrate that the cycle time is mostly stable (only very little deviations occur) for the IDs which are sent every (i) 20, (ii) 25, (iii) 50, (iv) 100, and (v) 500 ms. On the other hand, as observed in part (vi) of Figure 6.3 the multi-frame messages have a distinctive behavior compared to the others. The multi-frame messages are sent at larger time intervals, i.e., 5 s, and are followed by Data Transfer frames.

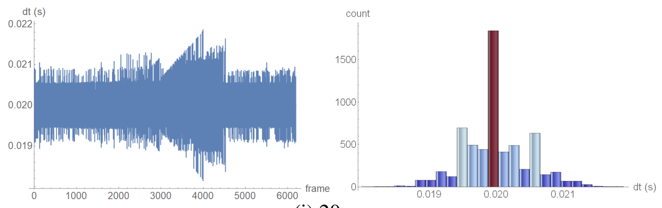
Nevertheless, it is important to draw attention to the time intervals between CAN messages in the full trace, i.e, all CAN IDs, given the fact that the security mechanism has to operate within these time restrictions. The left side of Figure 6.4 depicts the time interval between successive frames for the first 1000 messages from the recorded CAN traffic (to prevent overloading the figure). Their histogram distribution is shown in the right side of the Figure 6.4 for all captured frames. The minimum delay between successive frames is around 500 μs, which is approximately the amount of time that a CAN frame sent at 250 $Kbps$ spends on the bus. Ideally, this indicates that in order to prepare the ECU and make it available for the next frame transmission, any applied security mechanism must not exceed an execution time of 500 μs.

6.2 Proposed mitigation mechanism

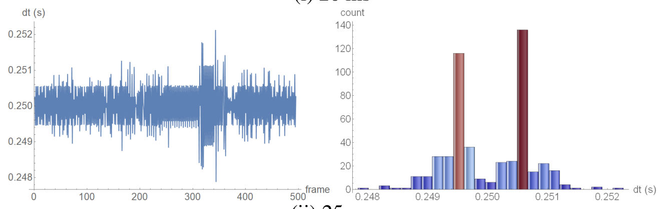
This sections gives an overview of the adversary model that is considered for current evaluation and then presents the complete solution designed for securing J1939 CAN bus communications.

6.2.1 Adversary model

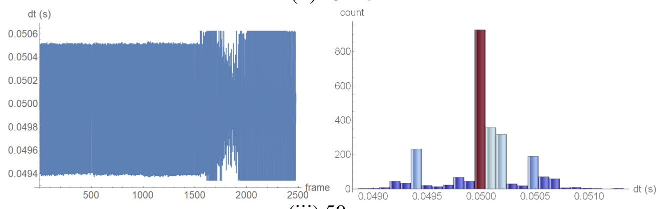
Replay, modification as well as DoS (Denial of Service) attacks are the common adversarial interventions that are taken into account while designing, developing and testing intrusion detection systems for CAN buses [64], [125], [47]. Note that, despite the fact that some



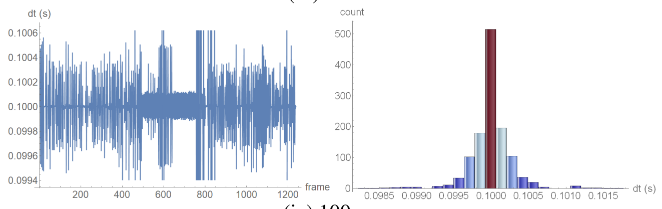
(i) 20 ms



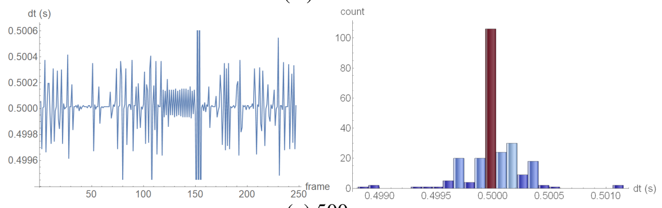
(ii) 25 ms



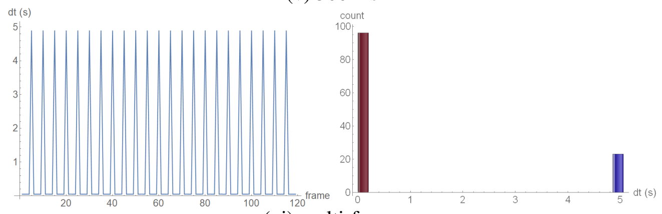
(iii) 50 ms



(iv) 100 ms



(v) 500 ms



(vi) multi-frame

Figure 6.3: Measured inter-frame delays as well as their histogram distribution for IDs with a cylcle time of: 20, 25, 50, 100, 500 ms and a multi-packet transmission

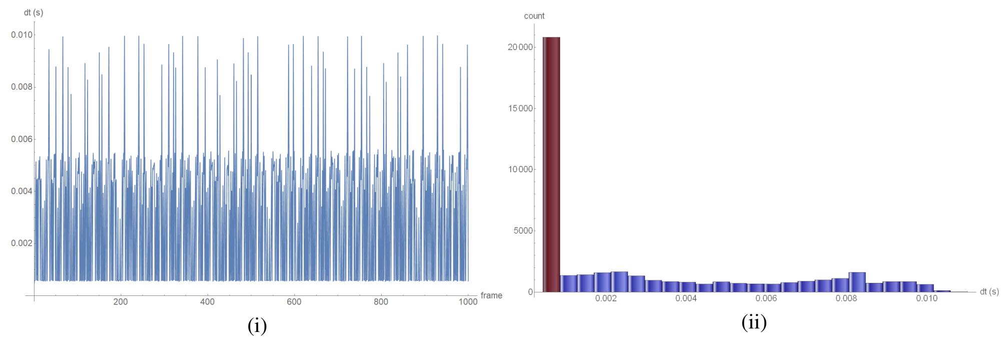


Figure 6.4: The accounted latency for the first 1000 messages as well as the histogram distribution with the recorded latency for all captured CAN messages

works consider spoofing attacks in the adversary model, such adversarial actions are in fact replays or modifications of frames with IDs that are part of the legitimate traffic. Furthermore, other papers have explored fuzzying attacks [116], in which frames with randomly generated IDs are injected on the bus. A part of these attacks accounts for frames with IDs that are not associated with genuine communication. The detection of such attacks is straightforward and can be done by simply evaluating whether the ID is part of the genuine traffic. When the generated IDs are part of the legitimate network communication, fuzzing attacks are in fact the modification attacks addressed by our work. Otherwise, as stated, they can be immediately detected since they use IDs that do not exist in the legitimate traffic.

The evaluation of the IDS will be based on an adversary model that includes both replay and modification attacks. From the IDS perspective, the implementation that relies on encrypted addresses allows for detect replays while the range checks that take advantage of the avalanche effect produced in the encrypted data-field enable the detection of modification attacks. This evaluation does not consider DoS attacks that can be done by injecting additional high priority frames on the bus, or by targeting the CAN frame format and the error confinement mechanism of the legitimate frames because they cannot be stopped considering the nature of the CAN bus (ID based arbitration). According to related publications, such as [121] and [17], an adversary can always destroy frames and inject messages with high priority IDs to flood the bus. Even if, monitoring the busload would be sufficient to detect DoS attacks that involve flooding, the ID oriented arbitration employed in CAN buses makes the IDS to defend against such attacks. There is no other way proposed in the literature to stop such attacks, except for decoupling sectors of the bus which was only very recently addressed in [119] and [120]. Targeted DoS attacks, such as the one proved in [121], in which the content of the frame is manipulated by an adversary to produce transmission errors and put nodes into a bus-off state, are much more difficult to detect and call for modifications in the CAN error handling mechanism that can not be implemented into an IDS.

6.2.2 Description of the proposed solution

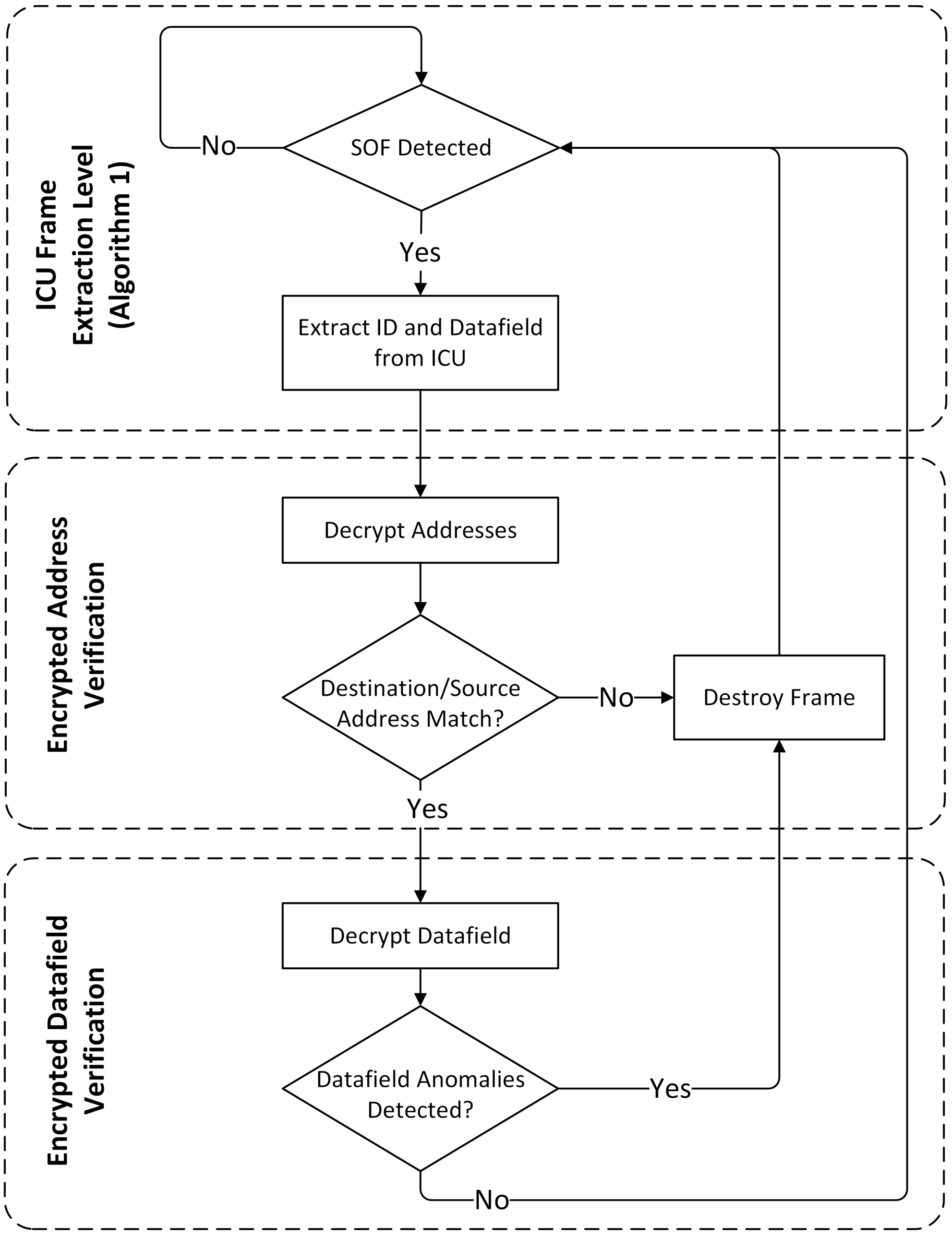

ICU Frame Extraction Level (Algorithm 1)
SOF Detected
No
Yes
Extract ID and Datafield from ICU
Encrypted Address Verification
Decrypt Addresses
Destination/Source Address Match?
No
Destroy Frame
Yes
Encrypted Datafield Verification
Decrypt Datafield
Datafield Anomalies Detected?
Yes
No

Figure 6.5: The two-stage IDS: encrypted addresses validation and payload anomaly verification

The envisioned solution relies on an intrusion detection and prevention system with two layers that evaluates the authenticity of the source and destination addresses in the first step and then monitors for abnormalities inside the payload. In order to accomplish this, the source and destination addresses are remapped with encrypted values that can be only verified by the genuine nodes within the network. Furthermore, the data field can be encrypted because doing so (together with the encrypted source and destination address of the ID) will prevent an adversary from deducing the layout of the payload. So, despite the existing specifications in the J1939 standard related to the parameters packed inside the frame content, the encrypted parts of the ID combined with the encrypted payload will confuse the adversary regarding the meaning of the parameters carried by the frames. In addition, this will make it easier for the IDS to detect malicious alterations inside the payload. The block diagram of the proposed two layer intrusion detection and prevention system is depicted in Figure 6.5. In accordance with the schematic from Figure 6.5, the ICU starts building the message by applying Algorithm 1 once a start of frame bit occurs, that is, a change from the recessive state to the dominant state. This is required to get the content of the ID and payload before the frame becomes available in the CAN buffer because once that happens, the message is already accessible to network nodes and cannot be discarded with an error flag. Subsequently, the proposed solution proceeds with the two stages of intrusion detection, which rely on two different mechanisms. These mechanisms are the usage of *encrypted addresses* and *data field encryption*, which are going to be discussed next. The detected adversarial frames that are regarded as intrusions by the IDS, will be destroyed with error flags.

Encrypted addressees – are used to substitute the typical address of both sender and receiver nodes assigned to each node with addresses that have been encrypted using a cryptographic one-way function. In this case, the encryption is done using AES. Both the sender and receiver node addresses are re-mapped at regular intervals using an address mapping matrix to prevent an attacker from injecting frames with disclosed addresses.

Integrating security elements in the ID field of a frame is consistent with recent works that address similar methods [126], [127], [128]. The current approach, however, unlike these works that aim to maintain the defined priority of the frames, encrypts only some portions of the ID field, i.e, the source and destination node addresses. We note that, according to J1939 specifications, only the first three bits of the ID, which remain unaltered in the current approach, are used to establish the priority of the frames.

The current solution exhibits 16 bits of security by correctly encrypting the address of the sender and receiver nodes that are part of the ID field, every one of them is 8 bits long. Obviously, this can be increased to 24 bits of security in order to meet the AUTOSAR standards [129] by encrypting the PDU format which is also part of the ID. Nevertheless, we consider that ID filtering should be still possible and at least some of the ID field has to be left unencrypted. This is the rationale behind staying with the 16 bit security level, which substantially limits the adversary's potential to successfully inject frames on the bus. Indeed, for the injection to have higher chances to succeed, the payload alterations must remain undetected.

Data field authentication using encryption – is an extra layer of security added by us, where the actual content of the payload is concealed using a fast encryption algorithm that relies on a lightweight block cipher. Moreover, this encryption sets room for an avalanche effect even if a single bit is manipulated by an adversary, facilitating the detection of attacks by appropriate range checks.

The experiments that follow demonstrate that by using SPECK [130], one of the fastest available block ciphers, high-end controllers can perform the decryption quickly enough to facilitate the real-time destruction of frames that were manipulated by an adversary. For well-motivated adversaries, encrypted traffic analysis could still represent an option. On the other hand, it should be exceedingly challenging to extract any relevant knowledge from the frames because both the identifier and payload are encrypted which makes them pseudorandom. A deeper examination of such an attack scenario is not possible for the current work due to space restrictions.

Key management. According to the suggested approach, each CAN node has to possess a symmetric secret key. In the current deployment, this key was initialized for each ECU with a randomly generated value that was hardcoded. Secure key sharing is a different topic which was investigated in recent works, and it would be out of scope to address mechanisms for this purpose. Because of this, we only draw attention to a few recent works that deal with similar challenges. A proposal that relies on the physical layer to securely exchange keys between the nodes from a CAN network and permits the grouping of nodes that share exactly the same keys, is explored by the authors of [131]. In a recent work [132], group keying with implicit certification using the elliptical curve Diffie-Hellman key agreement protocol was addressed. The elliptic curves are also taken into account by works [133] and [134] for key exchange between the CAN network nodes.

6.2.3 Unique encrypted addresses by ordered binary trees

One potential problem that can occur when random addresses are generated in order to replace the old ones, is related to the address collisions. According to the birthday paradox, there is a 50% chance of a collision if each pair (DA, SA) becomes substituted by a random 16-bit value after around $\sqrt{65536} = 256$ generated pairs of addresses. In order to avoid this, the mapping of the IDs was done using an ordered binary tree which solely stores unique values.

The construction of a binary tree is depicted in Figure 6.6. The nodes of the tree are added depending on their values. These values are the result of applying XOR between the last 16 bits of the IDs (8 bits for sender address and 8 bits for destination address) and address mask $\mathsf{msk}_{\mathsf{adr}}$. Now, an example is discussed focusing on the first node from the tree. The corresponding value for this node is 31463 which results from XOR operation between the initial ID `0xCFE4421` and $\mathsf{msk}_{\mathsf{adr}} = 3\text{EC}6$ as follows:

$$4421 \oplus 3\text{EC}6 = 7\text{AE}7 = 31463_{10} \tag{6.1}$$

The updated sender address in this instance is obviously `0xE7`, and the updated destination address is `0x7A`. In order to quickly examine for collisions in the generated addresses, the binary tree is ordered according to these resulting values. When a new value is created, the counters i and j are both incremented. The counter i represents a global counter that is incremented for each ID generation whereas counter j is a local counter that is incremented based on the value index for the actual ID. If a collision takes place, an additional value is generated and the global counter i is incremented. Although such a scenario is unlikely to happen frequently, it has to be accounted for.

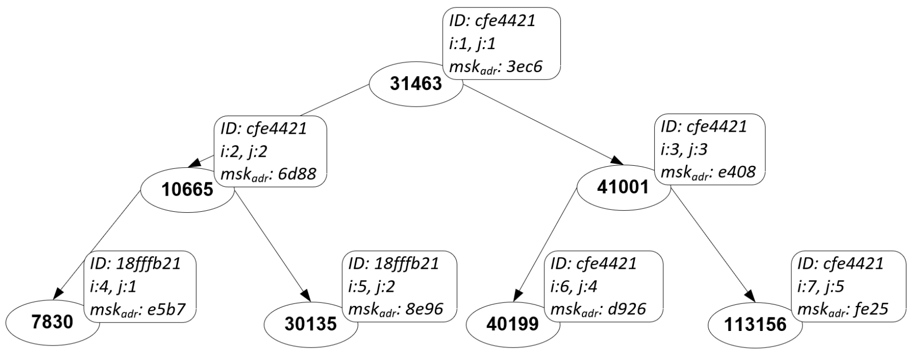


Figure 6.6: Generating unique addresses using an ordered binary tree

The address generation procedure based on the ordered binary tree must be executed periodically and comes up with concerns about processing power and storage capacity. Now, the formalization of these limitations and discussion on the trade-offs are provided. Let us consider the following:

- a collection of identifier-cycle pairs $\{(id_1, \delta_1), (id_2, \delta_2), ..., (id_n, \delta_n)\}$, that are figured out from the collected CAN traffic,
- the life-time of the ID masks tree Δ,
- the unicity time interval of the ID masks δ.

The δ parameter specifies the amount of time in which the generated addresses are new. In order to guarantee that all addresses are unique for every instance of the ordered binary tree, i.e, the same addresses are not used more times during Δ interval time, the ideal scenario would be $\delta = \Delta$, however, this may demand excessive amounts of memory and processing power. If these resources are not available to all nodes from the network, a $\delta < \Delta$ may be chosen. Because the majority of CAN messages are periodically transmitted by the nodes, it is easy to determine the ratio of unique encrypted addresses as follows:

$$\rho = \delta\Delta^{-1} \tag{6.2}$$

Considering these, the total number of ID masks λ_i that are created for each $id_i, i = 1..n$ and employed for the encryption of sender and destination addresses, can be defined as:

$$\lambda_{i\in\{1..n\}} = \rho\frac{\Delta}{\delta_i} = \frac{\delta}{\delta_i} \tag{6.3}$$

Moreover, the total number of ID masks Λ generated for all CAN traffic can be computed as follows:

$$\Lambda = \sum_{i=1}^{n} \lambda_i \tag{6.4}$$

The total number of encryption masks might be a bit greater than Λ because of potential collisions that can occur, which results in an average computing time of:

$$\mathbb{T}_{\textsf{comp}} = \left(\sum_{i=1}^{n} \frac{2^{16}}{2^{16} - i}\right) \times t_{crypt} \tag{6.5}$$

Here, t_{crypt} is the time needed for generating a set of encrypted destination and source addresses. Since one AES computation produces 128 random bits and only 16 bits are employed as mask in order to generate a pair of encrypted addresses, then t_{crypt} is computed as follows:

$$t_{crypt} = t_{AES}/8 \tag{6.6}$$

where t_{AES} is the time necessary for one AES encryption, which can be used as masks for 8 encrypted addresses. Given that there are only 41 IDs in the traffic collected from the target vehicle and typically less than 100 IDs on real-world buses, $\mathbb{T}_{\textsf{comp}}$ can be estimated for the actual practical needs as follows:

$$\mathbb{T}_{\textsf{comp}} \approx \Lambda \times t_{crypt} \tag{6.7}$$

Considering that, on average, only 1-2 additional encryptions with a probability of 0 collisions equal to:

$$\frac{2^{16}!}{(2^{16} - n)!2^{16n}} \tag{6.8}$$

are needed to generate 50–100 unique encrypted addresses in the 2^{16} address space, then the previous estimation is roughly equal to the real number. In this relation, if there are n IDs then 2^{16n} accounts for total number of potential addresses while $2^{16}!/(2^{16} - n)!$ denotes the number of unique addresses. In the experimental section, specific measurements that confirm these estimations will be presented. The memory cost for storing the tree is $\Lambda \times m_{node}$, where m_{node} is the memory space needed by each node in the tree. The CPU load can be simply represented as:

$$\mathbb{CPU}_{\mathsf{load}} = \mathbb{T}_{\mathsf{comp}} / \Delta \tag{6.9}$$

This is due to the fact that one such encrypted address tree must be generated for each time interval Δ.

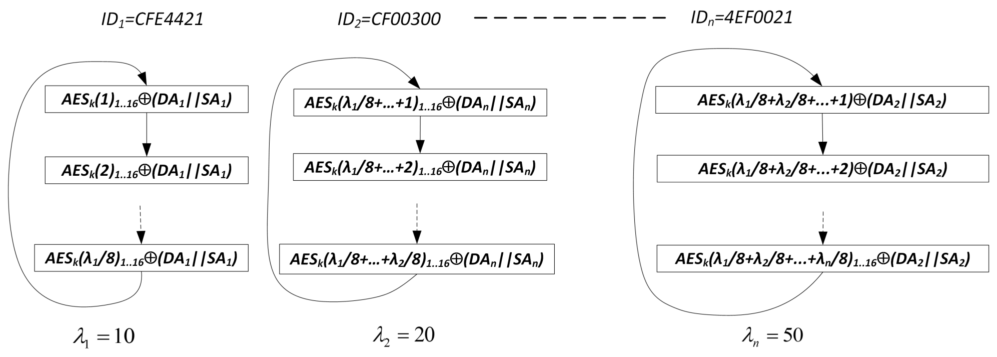

ID1=CFE4421
ID2=CF00300
IDn=4EF0021
AESk(1)1..16⊕(DA1||SA1)
AESk(2)1..16⊕(DA1||SA1)
AESk(λ1/8)1..16⊕(DA1||SA1)
AESk(λ1/8+...+1)1..16⊕(DAn||SAn)
AESk(λ1/8+...+2)1..16⊕(DAn||SAn)
AESk(λ1/8+...+λ2/8)1..16⊕(DAn||SAn)
AESk(λ1/8+λ2/8+...+1)⊕(DA2||SA2)
AESk(λ1/8+λ2/8+...+2)⊕(DA2||SA2)
AESk(λ1/8+λ2/8+...+λn/8)1..16⊕(DA2||SA2)
λ1 = 10
λ2 = 20
λn = 50

Figure 6.7: Re-mapping of sender and destination addresses using counter mode encryption as well as circular lists

Figure 6.7 provides a more detailed overview of how the sender and destination addresses map is exported to a circular list associated with each ID, considering $\Delta = 1s$ and coverage $\rho = 100\%$. The first ID is transmitted periodically at 100 ms, thus $\lambda_1 = 10$, the second and third have cycles of 50 ms and 20 ms, respectively, resulting in $\lambda_2 = 20$ and $\lambda_3 = 50$. The ID is shifted in accordance with the circular list throughout each transmission. In order to prevent an adversary from memorizing the re-mapped IDs, the ordered binary tree needs to be updated periodically using new AES computations. Additional insights on this, coupled with experimental evaluations, will be provided in the upcoming section, but in theory a new address tree might be generated every second or even more quickly.

In order to provide a clearer picture of these tradeoffs, in what follows, a concrete example based on the CAN network from the target vehicle, that is the subject of this investigation, is discussed. The CAN traffic recorded from the heavy-duty vehicle contains 33 cyclic IDs transmitted by three different nodes as follows: 2 IDs are sent by TCM, 10 by ECM as well as 21 by BCM. The remaining IDs, up to 41 are not periodically transmitted by nodes, thus it is employed a single encryption mask in these circumstances. Figure 6.8 shows the CPU load in percents accounted during the life-time $\Delta = 1\ s$ of the ID masks tree, with a coverage ρ ranging from 10% to 100%. In other words, when using a coverage of 10%, each ID will be repeated ten times, whereas for a coverage of

100%, every ID is used one time for a second. As one AES symmetric encryption takes between 24.8 μs and 2.8 ms on high-end boards and low-end boards, respectively, these computation requirements were chosen as a baseline for the minimum and maximum time needed for one encryption. The high-end device group exhibits a CPU load lower than 0.15% even when $\rho = 100\%$ while the low-end device group shows a CPU load of 17.92% which nevertheless remains reasonable. The experiments in the following section validate these estimates.

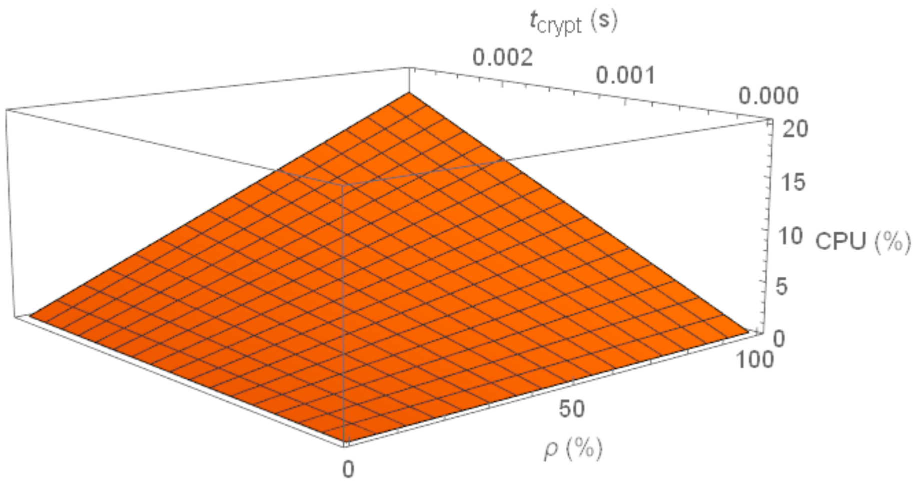


Figure 6.8: CPU load recorded during address generation procedure relative to the encryption time t_{crypt} as well as coverage ρ

6.3 Experimental evaluation

This section presents an experimental evaluation regarding the performance of the proposed IDS. Firstly, the computational results measured on automotive graded controllers are presented. Then, this section gives an overview of the active defense mechanism regarded as prevention system. The section ends with results on intrusion detection accuracy obtained for various J1939 specific parameters.

6.3.1 Computational results

In what follows, the effectiveness of the encryption operation is assessed. We do this because in-vehicle controllers are frequently used in real-time applications that depend on computational power and storage capacity. To demonstrate that the suggested method is computationally appropriate for automotive devices, i.e., in-vehicle controllers, the experiments were conducted to measure the runtime required for the construction algorithm of the ordered binary tree. In these experiments, a 32-bit S6J32GEKSN microcontroller from Cypress's Traveo Family is used as a reference for the high-end automotive graded controllers. This microcontroller has an ARM cortex R5 single core processor and can run at a top frequency of 240 MHz while the data and instruction cache are activated. There are 2048 KB of Flash and 128 KB of RAM available on the microcontroller. The current investigation also includes two other high-end representatives from the Infineon TriCore series. The first one is a TC1797 microcontroller, which has a TriCore V1.3.1 core clocked at a maximum speed of 180 MHz and offers 4 MB of Flash as well as 156 KB of RAM. The second one, TC397, relies on a more recent generation core that incorporates 6 cores and can operate at up to 300 MHz. It comes with 16 MB of Flash and 2528 KB of RAM. As a baseline for the low-performance device group, the evaluation was conducted on a 16-bit S12XF chip with 32 KB of RAM and 512 KB of Flash, which operates at up to 50 MHz.

The computational costs of SPECK with 64-bit block and 96-bit key and AES with 128 bit block and key algorithms used for the symmetric encryption are shown in Table 6.2 while the costs for the generation of the encrypted addresses using different lifetimes Δ and coverage ρ, are shown in Table 6.3. The measured execution time for the AES encryption is in the range of 24.8 μs - 77.6 μs when high-end devices are used (S6J32GEKSN, TC1797 and TC397). On the other hand, the computational time is less than 3 ms when the S12 was used, a representative for low-end boards. The run-times measured for the SPECK are 5-15 times faster for high-end boards and for low-end board, i.e, S12, exhibits roughly the same run-time like the one obtained for AES. Most probably, this is because the microcontroller relies on a 16-bit architecture, whereas the used code was designed for 32-bit systems and the 16-bit conversions are handled more efficiently (at the compiler level). Even in the scenario of full coverage, i.e., $\rho = 100\%$, the costs for the generation of the ordered binary tree ranges from 2.16 ms when a high-end board is employed to 246 ms in case of a low-end board. On the S12XF512 controller, in case of $\Delta = 10$ s, $\rho = 25\%$, the address generation procedure cannot be accommodated using a single RAM page and consumes roughly 50% of the RAM, making it excessively expensive for a single operation. In summary, the address generation procedure occupies less than 1% of the CPU time per second for high-end boards while for low-end board less than 25% of the CPU time per second, which is reasonable. We note that, for security concerns, the address binary tree should be re-generated at regular time intervals, such as every 1–10 s. Moreover, in order to prevent de-synchronizations, fresh addresses will be prepared prior to use them, and after older addresses have been consumed, the more recent ones will take their place. Therefore, the address refresh procedure necessitates two address trees, which would double the amount of storage space needed, but even at that cost, which varies from 0.86–7.47 KB, it is still manageable.

Table 6.2: Computational results for the symmetric encryption using AES and SPECK

	Encryption	
	AES 128/128	SPECK 64/96
Platform	CPU	CPU
S6J32GEKSN	$24.8\mu s$	$5.32\mu s$
S12XF512	$2.8ms$	$2.56ms$
Tricore TC1797	$77.6\mu s$	$5.30\mu s$
Tricore TC397	$28.672\mu s$	$6.97\mu s$

Table 6.3: Computational results for the generation of the ordered binary tree in relation to life-time Δ and coverage ρ

	Address Generation Procedure							
	$\Delta = 1s, \rho = 25\%$		$\Delta = 1s, \rho = 50\%$		$\Delta = 1s, \rho = 100\%$		$\Delta = 10s, \rho = 25\%$	
Platform	CPU	MEM	CPU	MEM	CPU	MEM	CPU	MEM
S6J32GEKSN	$588\mu s$	0.86KB	$1.05ms$	1.49KB	$2.16ms$	2.98KB	$5.84ms$	7.47KB
S12XF512	$68ms$		$120ms$		$246ms$		N/A	
Tricore TC1797	$1.76ms$		$2.98ms$		$6.24ms$		$15.20ms$	
Tricore TC397	$746\mu s$		$1.30ms$		$2.79ms$		$7.37ms$	

6.3.2 Active prevention mechanism

Figure 7.14 depicts the experimental setup that was used to validate the efficacy of the current approach. This includes the S6J32GEKSN development board that handles the CAN communication acquisition via the input capture unit (ICU) pin, a S12 board that plays the role of an adversary, a power supply, as well as a PicoScope for monitoring the CAN messages. To enable the real-time detection and elimination of attack frames, a software application that includes the following components was integrated on S6J32GEKSN board:

- the acquisition and verification mechanism of the captured CAN frames,
- the CAN frame destruction mechanism which is initiated when an attack frame is detected.

Next, these two components are explained.

CAN frames acquisition. Hardware-wise, the ICU pin as well as the actual CAN Rx pin, both regarded as inputs for the µC are connected to the Rx output from the CAN transceiver. Figure 6.9 shows the suggested hardware design while Figure 6.11 presents the software algorithm. In most cases, an ICU is used to determine the duration time of an input signal connected to the correlated pin of the microcontroller. The ICU may determine either period or level duration according to its configuration. The measurement

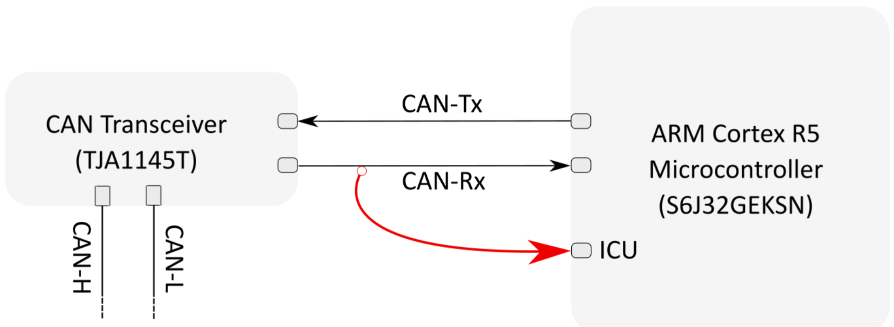


Figure 6.9: The hardware design for frame acquisition

of level duration is enabled by using a free-run timer (FRT) that is restarted using the flag *TReset* each time a level change occurs (like a trigger) as can be seen in line 4 from the algorithm.

Given this context, the employed algorithm relies on the ICU to determine every level duration during the reception of a CAN frame. In other words, it determines the number of bits, i.e, n, which are regarded either as dominant or recessive. To accomplish this, the acquired FRT value measured in ticks is divided by a value that represents the duration time (in ticks) of a single CAN bit which depends on the CAN baud rate. For the actual scenario the baud rate is 250 $Kbps$, i.e., 4 $\mu s/bit$, which corresponds to 240 ticks at a peripheral clock frequency of 60 MHz. This value, defined as Tbit is initialized in line 2 from the algorithm. To ensure that the entire value for one bit is not lost as a result of the integer division, half of the value for one bit, i.e, 120, is added before computing the number of bits as can be seen in line 3.

Also, in the same line 3 and in lines 6-9 of the algorithm, respectively, the CAN stuffing bits accounted in the acquisition buffers are removed. In case of the value calculated in line 3 is greater than the value equivalent with the duration time of five bits that has the same polarity, this value is discarded since it represents an inter-frame space. In this instance, the initializations necessary for a new SOF bit are carried out as well as the acquisition buffer indexes are accommodated to be ready for the following frame. These are done in lines 20-22. We note that *stuff* and *polarity* variables are re-initialized. Because these variables were globally declared, the RAM startup routine of the microcontroller will set them to 0. Also, line 22 shows how many bits are reserved, i.e., 128 bits (4*32), in the acquisition buffer for every captured CAN message.

A pointer *p is used to access the acquisition buffer, which is then filled bit by bit (lines 12–14) in accordance with the number of bits that have the same polarity computed in line 3. As shown in line 19, the polarity is switched for the next captured bits. We note that the global variable msg serves as a frame counter inside the buffer while the $step$ variable denotes the bit index within the same buffer. When the application starts, these

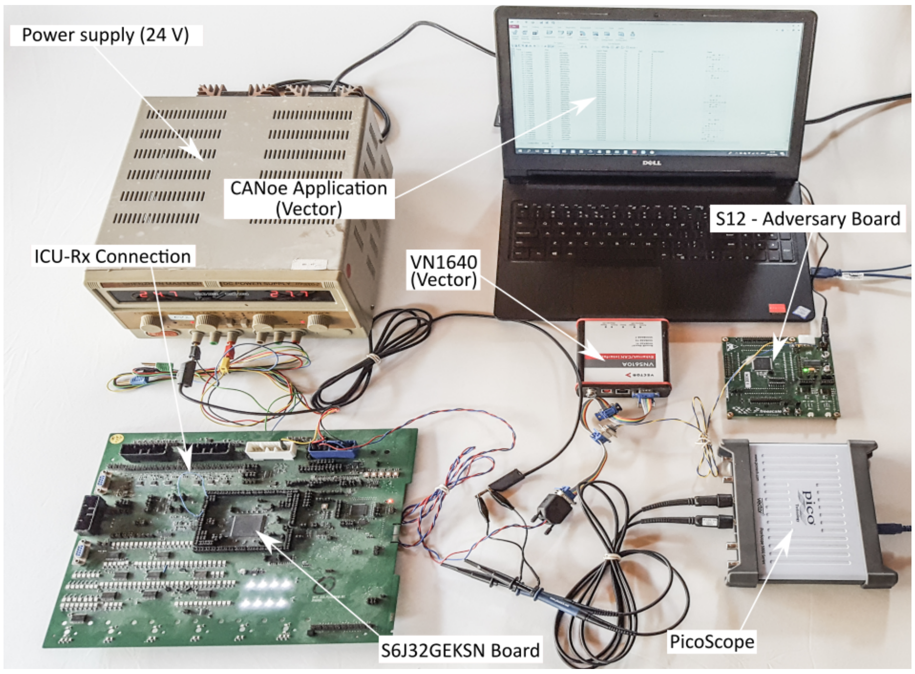


Figure 6.10: Experimental setup with the VN1640 device, S6J32GEKSN and S12 boards, Power Supply as well as the PicoScope for monitoring the CAN communication

variables are initialized, and as soon as the buffer is fulfilled they are reset. Specifically, the current application uses a buffer size of 128 CAN messages.

CAN frame destruction. The ICU detection stops working as soon as the complete payload of the frame has been recorded in the acquisition buffer. This means that 103 bits, as shown in line 15 from the algorithm, are stored in the acquisition buffer corresponding to the following fields of the CAN frame: SOF (1 bit), arbitration (32 bits), control (6 bits) as well as the data field (64 bits). The ICU detection is turned off so that the buffer can be processed without further interruptions from the ICU that could result from level changes triggered by the remaining CAN frame fields, such as the CRC, ACK, etc., that are received. In the first step, the legitimacy of the ID is verified. This is done by checking if the ID of the captured frame matches the one from the precomputed set of IDs that is stored in the memory. The precomputed set of IDs is stored in a matrix and generated based on AES symmetric encryption, as was previously mentioned. The equality verification with the ID from the captured frame is done based on an index that stands for the matrix column. In the instance that the IDs match, the index is incremented. The same index is restarted, i.e, set to 0, when it reaches its maximum value, which is equal to the total number of precomputed encrypted addresses for the corresponding ID. In contrast, if the IDs do not match, the procedure for eliminating the frame to prevent that the rest of nodes do not

Algorithm 1 CAN frame acquisition

```
**Input:** *TValue* (current value of the timer in ticks)
**Output:** *frame* (content of the ID and datafield)
procedure FRAME ACQUISITION
    $Tbit \leftarrow 240$
    $n \leftarrow (TValue + 120)/Tbit - stuff$
    $TReset \leftarrow 1$
    if $(n \leq 5)$ then
        if $(stuff = 1)\&\&(n = 4)$ then
            $stuff \leftarrow 1$
        else
            $stuff \leftarrow n/5$
        end if
        for $i = 0, i \leq n$ do
            $p \leftarrow \&var[step/32];$
            $*p \leftarrow (*p) \vee ((polarity) \ll (31 - (step\%32)))$
            $step \leftarrow step + 1$
            if $(step \geq (((msg - 1) \ll 7) + 103))$ then
                return $frame$
            end if
        end for
        $polarity \leftarrow !polarity$
    else
        $polarity \leftarrow 0, stuff \leftarrow 0$
        $step \leftarrow (msg - 1) * 128$
    end if
end procedure
```

Figure 6.11: CAN frame acquisition algorithm

receive it, is triggered. The frame is destroyed by holding the Tx pin in the dominant state for a time interval greater than the time required for the tranmision of 5 CAN bits. This will lead to a stuffing error on the CAN bus. Figure 6.12 shows the procedure of destroying CAN frames. This was captured using a PicoScope tool, configured for CAN serial decoding, that monitors both the CAN-H and CAN-L lines. The activation of the error flag causes the decoding of the CAN frame to be interrupted as can also be seen in the right part of Figure 6.12. For enabling the forcing of Tx CAN pin to a dominant state, the pin function was configured as GPIO (General Purpose Input Output), while the direction must be set as output. Afterwards, the pin value is configured to 0 logic. The ICU detection is resumed as soon as the intrusive frame is eliminated In order to be ready for the subsequent CAN frame.

6.3.3 Detection of intrusions using J1939 specifications

In what follows, we describe the detection mechanism of malicious modifications inside the payload. This is based on certain knowledge that may be extracted from the predefined structure of J1939 frames according to J1939 standardization. For brevity, the investigation concentrates on the following five parameters:

- engine speed measured in rpm,

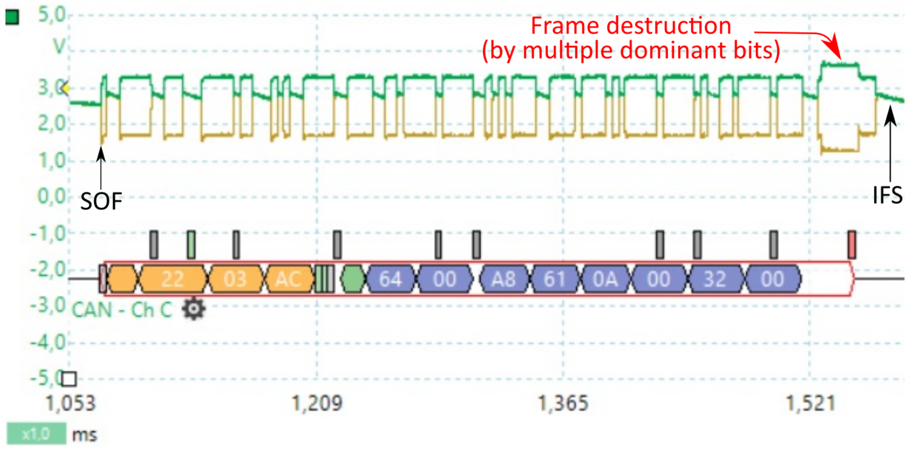


Figure 6.12: Digital oscilloscope plot with CAN frame destruction

- engine torque measured in $\%$,
- fuel consumption measured in l/h,
- vehicle speed measured in km/h,
- engine temperature measured in $^\circ C$.

Certainly, the evaluation can be expanded to encompass additional parameters carried by the frames. The engine speed and engine torque are transmitted at a periodicity of 20 ms by a frame with ID `0xCF00400`. The fuel consumption is carried by a message with ID `0x18FEF200` while the vehicle speed corresponds to ID `0x18FEF121`, both of them are sent with a cycle time of 100 ms. The last parameter, i.e., engine temperature is sent more rarely, i.e., 1 s cycle time, and is associated with ID `0x18FEEE00`.

Table 6.4: Recorded variations for engine speed (rpm), engine torque ($\%$), fuel consumption (l/h), vehicle speed (km/h) and engine temperature ($^\circ C$)

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

The evolution of these signals throughout a 20 minutes period of regular vehicle operation is depicted (with blue line) in the left side of Figure 6.13. The right side of

Figure 6.13 highlights the fluctuation of differences between successive parameter values using a histogram distribution. We observe that despite the large data range, for example in the actual scenario, up to 2000 rpm, 20 l/h, or 35 km/h at a resolution of up to 16 bits, the fluctuation between the values carried by consecutive frames are rather minimal because of the high acquisition rate. For each parameter value $v_i, i = 1..n$, a vector that contains the differences accounted during regular operation is created as follows:

$$\Delta(v_i) = v_i - v_{i-1}, i = 2..n \tag{6.10}$$

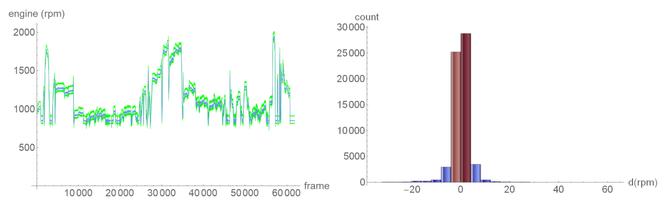
Let Δ_{min} and Δ_{max} represent the minimum as well as the maximum values recorded for this vector. The concrete min-max values can be seen in Table 6.4. Afterwards, based on these, the following verification $v_i \in [v_{i-1} - \Delta_{min}, v_{i-1} + \Delta_{max}]$ is performed for each newly recorded parameter value v_i.

The anticipated boundaries, represented by the minimum and maximum values, computed over the actual parameter values are displayed using a green line on the left side of Figure 6.13. This range checking allows the intrusion detection system to quickly detect the anomalies inside the payload. It is worth mentioning that, unlike the case of engine speed, engine torque, vehicle speed and coolant temperature, where both the maximum and minimum value of the range can be examined, for fuel consumption only the maximum value of the range may be monitored because, after a CAN traffic investigation, we remark that the minimum value, i.e, zero, regardless of the current value, is frequently provided.

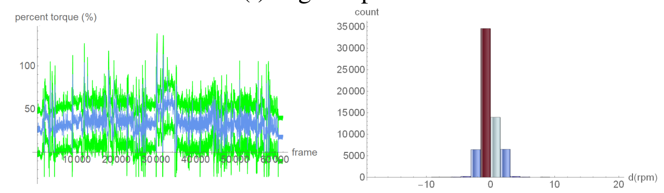
6.3.4 Accuracy results on detecting intrusions

By rebuilding the CAN bus topology from the target vehicle inside a CANoe based simulation, our evaluation relies on a realistic laboratory testbed. This is used for the performance evaluation of the proposed intrusion detection and prevention system. Moreover, the testing of the prevention mechanism can be done in this setting without harming the actual vehicle. Using this environment, the building of traces that are augmented by replay and modification injections is performed.

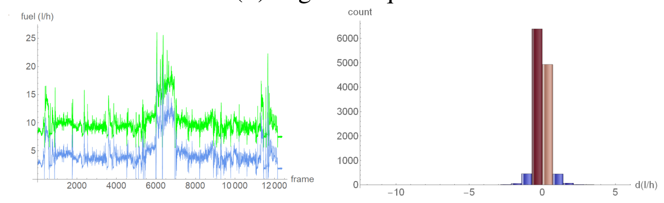
Both of these adversarial manipulations can be detected by the first stage which relies on the usage of encrypted addresses. Practically, since for each CAN ID the sender and destination addresses are replaced by the encrypted ones, the likelihood of guessing the proper future value for an ID is 2^{-16} (8 bits are used for each address). This is the case when one certain ID is targeted by an adversary. On the other hand, if the attack is not targeted on one particular ID, the most appropriate scenario for an adversary to succeed is when injecting frames with the start of the IDs that have the highest frequency, in this case, `0x18FE` is associated with 12 IDs. For such adversarial scenario, matching any of the 12 IDs which have the exact identical start is opportune. Therefore, in this circumstance, the guessing of the 16 encrypted bits will result in a cumulative probability equal with $12 \times 2^{-16} = 0.018\%$. This corresponds to a security level of around $-\log_2(12/2^{16}) = 12.41$ bits. The payload encryption, which is the second detection



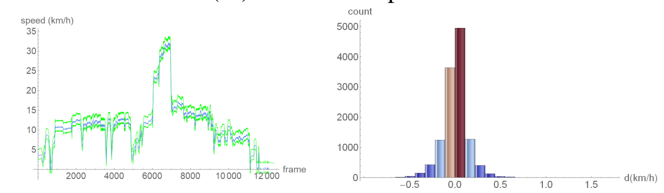
(i) engine speed



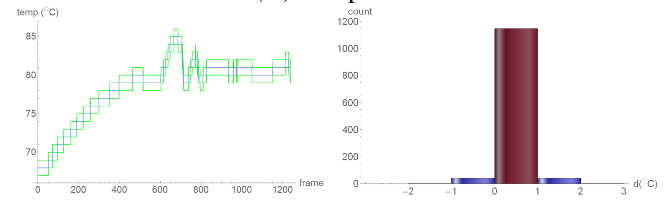
(ii) engine torque



(iii) fuel consumption



(iv) car speed



(v) coolant temperature

Figure 6.13: Predicted minimum and maximum limits accounted for (i) engine speed (rpm), (ii) engine torque (%), (iii) fuel consumption (l/h), (iv) vehicle speed (km/h) and (v) coolant temperature (°C) (left side) as well as histogram distribution of fluctuations between consecutive parameter values (right side)

stage, provides extra resistance to adversarial interventions even when single bits are manipulated because of the avalanche effect. After that, the range checks performed on every J1939 parameter of the payload will disclose that the frame was altered by an

Table 6.5: Detection results for attacks targeted on various J1939 parameters

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

adversary.

Table 6.5 presents the performance results in detecting intrusions. For a clearer picture, the corresponding security level is specified in the table as:

$$\ell = -\log_2(\text{FNR}) \tag{6.11}$$

Initially, the evaluation was conducted to see how the first stage of the proposed IDS that relies on *encrypted addressees* defends against attacks. For this, random addresses were generated to evaluate the possibility that an adversary will successfully guess any legitimate ID that is predefined in the encrypted address matrix. In this scenario, there is a very small possibility that an intrusion will go unreported, i.e., 0.01%, translating in a security level of 11.57 bits. This is roughly equal with the theoretical approximation. In case of attacks that target the payload alteration, the results show an excellent detection rate, i.e, higher than 97%, with the exception of engine torque. For this parameter the TPR exhibits a lower detection rate that is roughly 80%. The reason behind this is brought on by the fluctuation accounted for this 8-bit parameter (-28, 22), which is much larger than the ones recorded for the rest of the evaluated parameters, and due to this, the probability for an adversary to inject a value into this interval is higher. Even though the engine speed signal exhibits a wide range (-35,62), the adversary chances are significantly reduced because of the 16-bit signal length. Since there are no false positives (FPR=0%), all legitimate frames with valid addresses are correctly classified.

In scenarios where both IDS stages are employed, the probability of an intrusion to succeed is so negligible and, thus, it is not quantifiable by attack experiments. In other words, the probability that an adversary frame will remain undetected is in the range of 0.000159–0.006820%, which led to 0 undetected intrusions if the results are computed over this trace. Table 6.6 presents the synthetic computations of the detection rates obtained for the merged stages. These results are obtained by multiplying both FNRs, the first obtained from the attacks that target ID while the second from data field attack scenarios. This was done in order to account for the probability that the adversary ID will remain undetected as well as the probability that the manipulation of the datafield

Table 6.6: Estimated detection results for combined layers: ID and data field

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

will remain undetected. The accuracy of the results presented in Table 6.6 indisputably demonstrate that the probability of an adversary to mount an undetected attack is extremely low. None of the reported attack messages will be properly received by the nodes from the network thanks to the frame destruction technique that was introduced in a prior section.

Overall security level. The proposed IDS exhibits a security level that meets or potentially exceeds the AUTOSAR SecOC [129] requirements. This standard calls for packing inside each message 24-28 bits as an authentication tag as well as 0-8 bits regarded as freshness parameter. Practically, the fresh addresses that are predefined in circular lists exceed the 0-8 bit freshness threshold.

The proposed IDS, according to already described computations, obtains 12 bits of security based on the encrypted addresses. The reason behind using only 16 bits from the ID (sender and destination address) is to facilitate the message filtering at the CAN driver level by utilizing the rest of the bits from the ID, i.e, 13 bits. If that is not required and for the arbitration purpose only the first 3 bits from the identifier are reserved, all the rest of 26 bits from the ID can be encrypted, yielding a security level of:

$$-\log_2(41/2^{26}) \approx 20 \ bits. \tag{6.12}$$

This computation was done assuming that the collected traffic contains 41 message IDs. Regardless of whether this is not the case, each parameter packed inside the payload provides an extra 2–8 bits of security, as can be seen in Table 6.5. In the worst circumstance, this results in a rough security level of 2 bits/byte, as exhibited by the engine torque parameter, for which was obtained the lowest accuracy in detecting intrusions. We note that, constant parts of the payload provide an even higher security level of 4 bits/byte considering that the avalanche effect will lead to modification of minimum 50% of bits. In the light of the above, at 2 bits/byte, it is anticipated that decrypting the payload will result in a corresponding security level of a minimum 16 bits for each message. The security level of 28 bits that is obtained by merging the 16 bits from the payload with the 12 bits from the ID field fully satisfies the 24-28 bit range proposed in AUTOSAR SecOC requirements [129]. Therefore, the suggested IDS works well and complies with the actual security requirements. In comparison to other IDS solutions that do not rely on encrypted addresses as well as on the avalanche effect of the payloads, such as those in

[96], [59], [135], [64], etc., the proposed IDS is obviously much more secure.

6.4 Concluding remarks

As proven by the experiments, the proposed mitigation approach, which is specially designed to comply with J1939 requirements, shows an effective way to detect as well as to eliminate the attack frames in real-time. Since the payloads are fully allocated in J1939 specific implementations, the truncated MACs required by AUTOSAR specifications [129] cannot be accommodated inside them. To circumvent this issue, the current solution relies on encrypted addresses as well as symmetric encryption on the data field. The first one will make it more difficult for an adversary to determine the function of each CAN packet and to inject attack frames with proper IDs on the bus, while the second one will produce an avalanche effect if the encrypted payload is altered (which sets room for proper range checks that will spot the attacks frames even if a single bit was altered inside them). Moreover, as demonstrated in the experimental section, the proposed solution comes with reduced computational costs and is suitable for deployment even on low-end development boards. Last but not least, the proposed solution takes advantage of the faster reconstruction of the CAN frames content provided by the ICU to instantly eliminate intrusion frames. Future work could involve the extension of this evaluation on J1939 CAN-FD networks for which the specifications were recently released under J1939-17 [91] and the use of cryptographic hardware accelerators from the development boards to improve the execution time for security computations.

Chapter 7

Intrusion detection for control systems on SAE J1939 CAN buses

The content of this chapter is based on the results of the author published in [23]. This chapter explores adversarial tactics as well as potential defense mechanisms at the control system level in the context of J1939 heavy-duty vehicle buses. This low-level strategy complements the commonly used CAN bus attacks, in which a more knowledgeable adversary can bypass the detection. We also propose appropriate solutions to counter such attacks. In fact, as we further demonstrate in the experimental section, current methods based on machine learning algorithms will mainly fail to spot such adversarial manipulations. The experiments are conducted using a setup that connects the Simulink environment (an extension of the MATLAB platform) with the CANoe environment. The first environment is responsible with the simulation of in-vehicle control systems, while the second allows for the simulation of in-vehicle networks.

7.1 CANoe-Simulink integration

This section provides an overview of CANoe environment and its interaction with the Simulink environment. In-vehicle networks can be designed, simulated, analysed as well as tested using CANoe, a market-leading software development solution. Additionally, CANoe offers support for the connection with other widely used tools in the industry, such as Simulink, a MATLAB-based environment employed in the modeling of dynamical systems, e.g., in-vehicle control systems. Furthermore, Simulink allows C code generation and this code can be effectively ported on automotive graded platforms.

Figure 7.1 shows a typical CANoe simulation setup. In the right side, a simulated bus from the CANoe environment is depicted. The CAN communication from the simulated environment can be transmitted on the actual physical bus, which is represented in the left side, by using hardware components produced by Vector, e.g., CAN-case, VN1600,

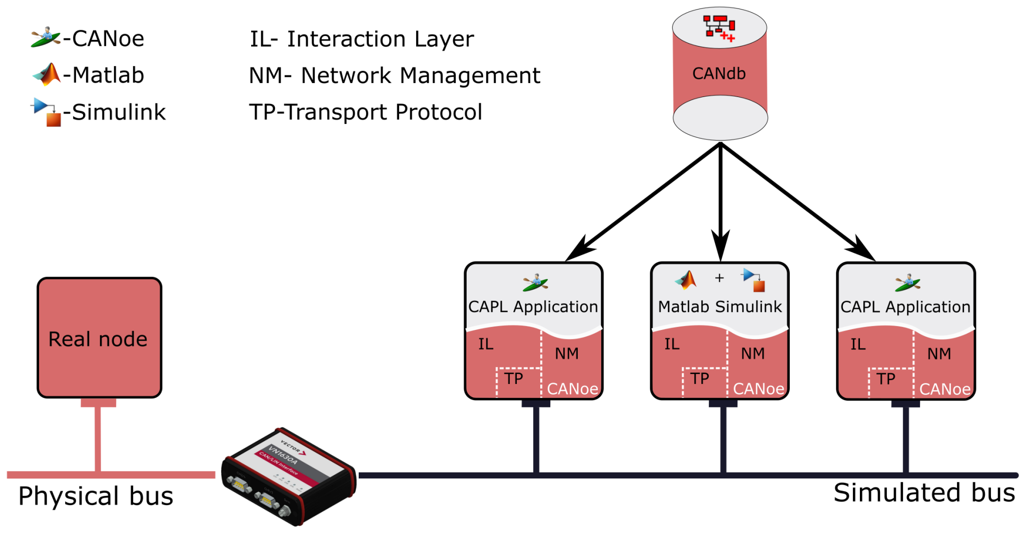


Figure 7.1: Typical CANoe simulation setup

etc. In this particular scenario, there are three CAN nodes in the simulation. In order to define the behavior of a CAN node, multiple software layers are used. The application layer is located on the upper side, and operates with the bus signals or the values obtained from various types of sensors or actuators. These values can be received as inputs via the graphical user interface. The CANoe environment employs CAPL (Communication Access Programming Language) based code for the deployment of the application layer. The event-controlled programming language CAPL has a syntax that is roughly comparable to the C language and is enhanced by a variety of particular functions which respond to the real-time events. These events can be related to CAN network communication (such as the reception of a CAN message, changing the value of a signal inside payload, etc.) to changes that could happen in the CANoe (like starting or stopping the simulation, etc.) or to timings (e.g., elapsing a timer value).

In the actual configuration, the CAPL code is utilized to simulate a real-world adversary that injects attack messages on the CAN bus. The interfacing with CAN messages and parameters, as well as with values retrieved from sensors, becomes easier by utilizing a CAN database (CANdb), which saves these as distinct objects: CAN signals, CAN frames, environment variables. This makes them easily accessible in the CAPL based programming. Moreover, the CANoe environment permits the extension of the CAPL application layer or even its total replacement with Simulink models (a capability that is in fact employed in this chapter).

The interaction between both the application and physical layer is mediated by the next three layers:

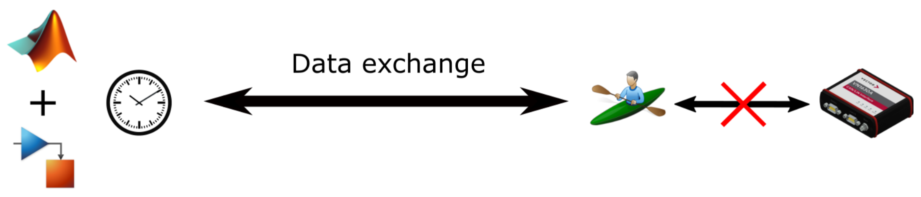
- interaction layer (IL),
- network management layer (NM),
- transport protocol layer (TP).

A brief summary of each is given in the paragraphs that follow. The IL layer is responsible for the interfacing between the application layer and the low level drivers. Since IL receives data stream by is in an application-level format (bus signals), it is his responsibility to ensure the conversion to a bit or byte format that is compatible with the physical layer. The IL not only converts the bus signals, but also assigns them to a particular CAN frame and manages their transmission on the bus. The NM layer, which carries out various bus management tasks, enables the optimal operation of a CAN network. Some examples of these tasks are:

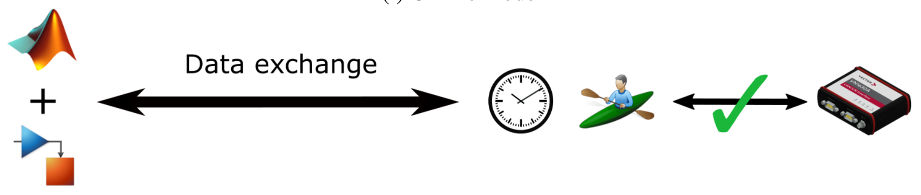
- recognizing the CAN nodes at power on sequence,
- monitoring the CAN bus during operation phase,
- coordinating the change from one state to another (the sleep state),
- providing information regarding the status of the CAN network.

The TP is responsible for handling the layout of the data link layer. Certain scenarios call for the transmission of data over CAN that cannot be carried by a single CAN frame as is greater than 8 bytes (the maximum payload size allowed by standard CAN). Such examples include the use of diagnostic buffers and multi-frame messages. Consequently, the data must be split into many frames and re-packed on the receiver side. This is done by the TP.

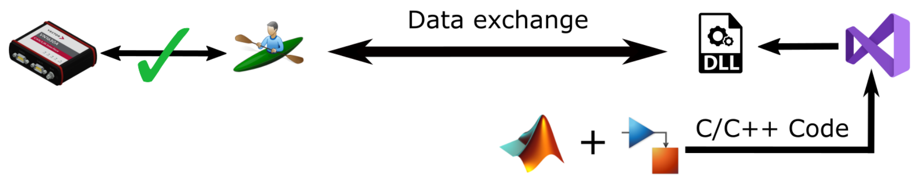
As shown in Figure 7.2, there are three different modes that ensure the connection between Simulink and CANoe during run-time. The first is the offline mode and is represented in Figure 7.2 (i). In this mode, the simulation operates inside the Simulink environment, which serves as the lead system, whereas the CANoe environment acts as a subordinate system. The offline mode, which is primarily employed for debugging needs, relies on the simulation time-base handled by Simulink. Therefore, a real-time simulation cannot operates in this mode, and no hardware device interaction is possible. Like in the case of offline mode, the Simulink environment operates as the host for the simulation also in synchronized mode. This is depicted in Figure 7.2 (ii). The main difference between this mode and the previous one is regarding the simulation time-base, which now comes from the CANoe environment. As a result, the simulation runs in real-time and enables the interaction with Vector hardware devices. The hardware-in-the-loop mode, which exhibits a different operation from the first two modes, is shown in Figure 7.2 (iii). In this configuration, the simulation is hosted by the CANoe environment. The Simulink models are embed into the CANoe simulation via `DLLs` (Dynamic Link Libraries). These



(i) Offline mode



(ii) Synchronize mode



(iii) Hardware in the loop mode

Figure 7.2: CANoe-Simulink interraction – operation modes

`DLLs` are the result of a build procedure conducted in the Visual Studio using C code produced by the Simulink Coder. In this study, the synchronized mode is used since this accommodates the real-time limitations and enables immediate modifications of the Simulink models and their testing as well, without the necessity to create a new DLL each time the models are modified.

7.2 Control systems and adversary models

This section gives an overview of J1939 simulation at the control system level and introduces the addressed adversary model.

7.2.1 J1939 simulation and Simulink models

The evaluation from this chapter focuses on the J1939 simulation configuration designed for a heavy-duty vehicle, which is made available by the CANoe tool [1]. The simulation setup is presented in Figure 7.3. Inside the CANoe environment, this simulation includes the defined CAN networks, together with the associated databases and nodes.

[1]https://www.vector.com/int/en/products/products-a-z/software/canoe

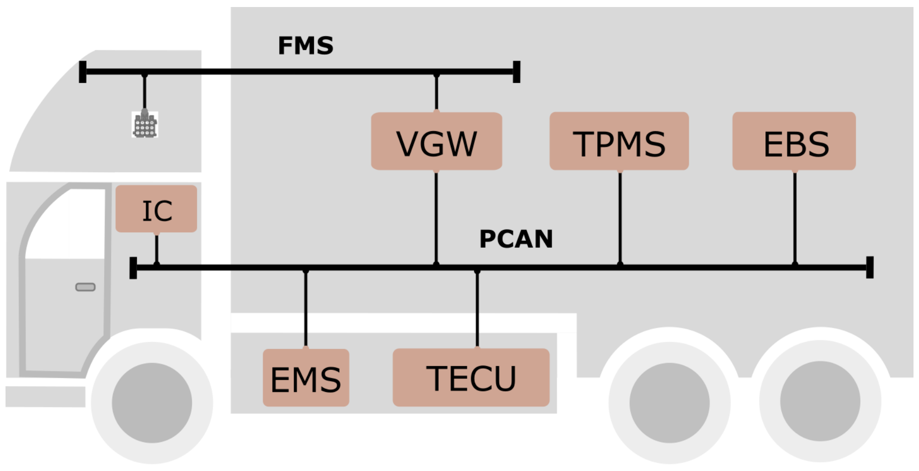


Figure 7.3: Common J1939 CAN network inside a heavy-duty vehicle

The J1939 network architecture accounts for two CAN buses, one of which is dedicated to Powertrain operations and the other to Fleet Management System (FMS) functionality. There are six ECUs on the Powertrain bus, one of them serves as a gateway and is also connected to the FMS bus. Table 7.1 lists the primary function of the ECUs that are used in the current evaluation, along with the addresses and signals that are allocated to them. The signals transmitted by the ECUs are listed in the Tx signal column, while the signals that the ECUs receive are listed in the Rx signal column. Figure 7.4 depicts the signal exchange between the ECUs. There is only one ECU on the FMS bus, which is the same gateway ECU (VGW) accounted also for the Powertrain bus. The VGW ECU is responsible for selecting the Parameter Groups that comply with the FMS standard [136] as well as for forwarding them from one CAN bus to the other.

The J1939 simulation is complemented by the Simulink models that include the control systems. This enables the modeling of the adversarial behavior. To protect the CAN bus from adversarial manipulations, the signals that circulate over the CAN network need to be predicted. Next, it is shown how these signals are estimated as well as the corresponding Simulink blocks.

Vehicle speed prediction

The Simulink model that calculates the vehicle speed in km/h using the shaft vehicle speed in rpm, which is transmitted by the TECU controller, is shown in Figure 7.5. Mathematically, the following formula is used:

$$v = \frac{3600}{1000} \times r \times \frac{2 \times \pi}{60} \times v_{shaft}, \tag{7.1}$$

Table 7.1: Brief overview of the nodes used in the Powertrain network

Node	Addr.	Function	Tx signal	Rx signal
EMS	0x00	Engine management system	Vehicle Speed, Trip Distance, Engine Speed, Torque	–
TECU	0x03	Transmission ECU	Shaft Speed	Clutch Slip, Gear, Torque
IC	0x17	Instrument cluster	–	Vehicle Speed, Shaft Speed, Trip Distance, Engine Speed,
VGW	0xE6	Vehicle gateway	–	Engine Speed

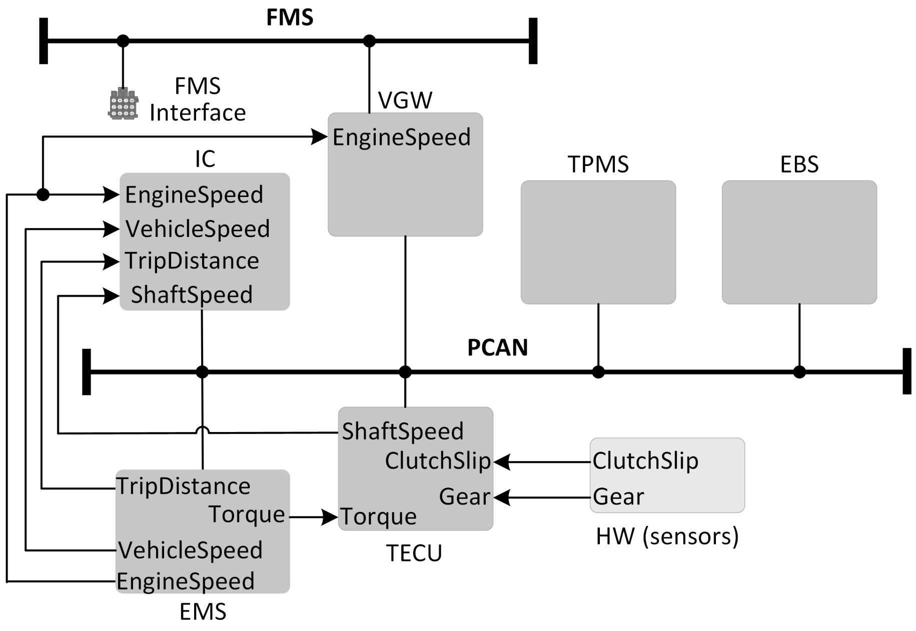


Figure 7.4: Signal circulation among ECUs on the PCAN bus

where v represents the vehicle speed measured in km/h, r is the shaft axle diameter measured in meters, i.e, 0.151 meters in this scenario, and v_{shaft} represents the shaft vehicle speed measured in rpm.

Typically, inside vehicles, a PWM signal that determines the rotation of the main axle is used by the TCU controller to calculate the shaft vehicle speed.

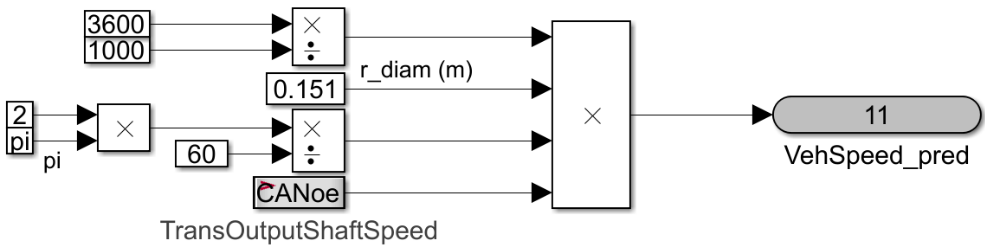


Figure 7.5: Vehicle speed prediction based on shaft speed

Trip distance prediction

Figure 7.6 illustrates how the trip distance is computed based on the vehicle speed. The following formula is used:

$$dist = \int \frac{v \times 0.1}{3600} \, dx, \tag{7.2}$$

where $dist$ stands for the trip distance, v represents the vehicle speed while the coefficient $\frac{0.1}{3600}$ is employed for converting the vehicle speed from km/h to m/s.

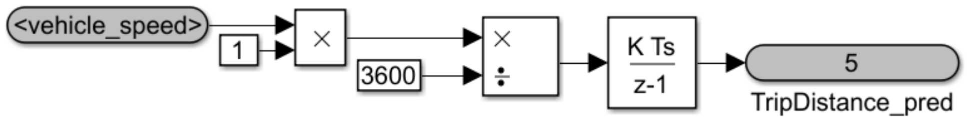


Figure 7.6: Trip distance prediction based on vehicle speed

Acceleration estimation

The computation of the acceleration depending on vehicle speed is presented in Figure 7.7. The following formula is used:

$$acc = \frac{\mathrm{d}v}{\mathrm{d}t}, \tag{7.3}$$

where acc stands for the acceleration in m/s^2, v represents the vehicle speed, and t is the time.

A low-pass filter was applied to the computed acceleration to remove unwanted spikes so that it could be used in the prediction of other vehicle signals, such as the engine speed and torque.

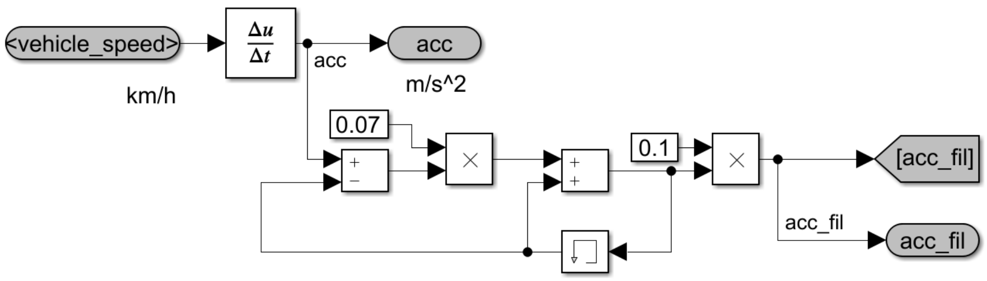


Figure 7.7: Acceleration estimation

Engine speed prediction

Figure 7.8 depicts the prediction of the engine speed using the filtered acceleration and gear. To do this, lookup tables are employed. These are frequently used in automotive and control systems based projects. In this scenario, the lookup tables are calibrated with breakpoints ranging from 3 to 100 based on a brief run-time of the simulation during which multiple gear shifts occur. With the help of these, the prediction of the engine speed is determined using the filtered acceleration and gear. The lookup tables are calibrated using the signals captured during the running of the CANoe simulation. This involves the selection of 3 to 100 values from the filtered acceleration for each gear and setting the corresponding value of the engine speed. We note that the interpolation is in fact used to calculate intermediary values. When needed, switching between two lookup tables depending on the clutch slip value is performed in order to increase the prediction accuracy. This is required since the engine speed ramps up or down more quickly when the clutch is engaged than it does when it is released (a case in which the signal variation is more stable). If the driver does not push the accelerator or brake pedals while the car is stationary, the estimated engine speed is configured to idle speed, which is 250 rpm.

Torque prediction

The engine torque is determined using lookup tables (similarly to how engine speed is estimated) that relies on the filtered values of acceleration as well as on gear, as can be seen in Figure 7.9. When needed, the switching between three lookup tables is performed. This requires to be done since if the accelerator pedal value is pressed more than 70% or if the power take-off module is turned on, the torque exhibits a different behavior. When the clutch is engaged, there is no torque request from the driver while the torque losses are significant and the value of the current torque is configured to the minimum value which is -1650 Nm and corresponds to the value from the CANoe simulation.

Engine speed signal as well as torque signal, are predicted without the use of a certain physical model and instead based on look-up tables that are calibrated using the behavior of the parameters deduced from previous runs of the J1939 CANoe simulation. Because

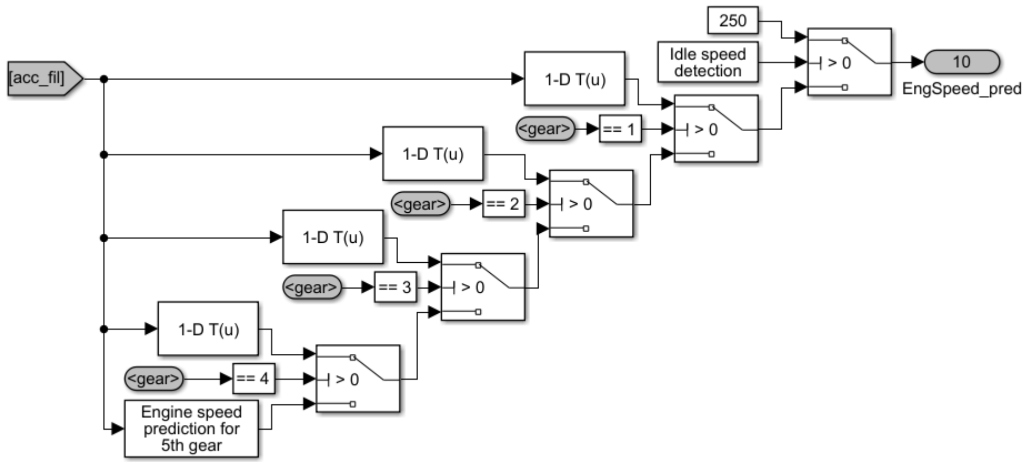


Figure 7.8: Engine speed prediction based on acceleration and gear

of this, the detection results from the experimental section will be poorer on these two signals, and should be regarded only as an example. More accurate predictions can be done by using improved models, but these are out of reach for the evaluation from this chapter.

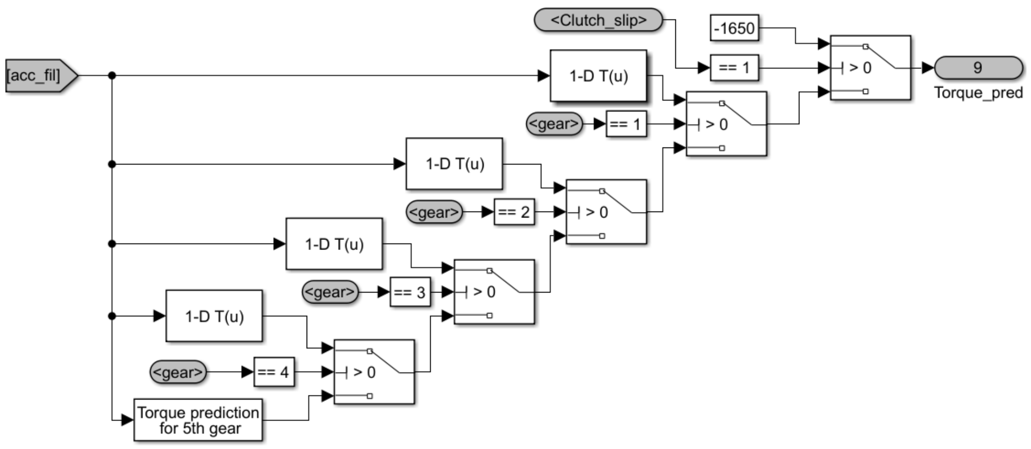


Figure 7.9: Torque prediction based on acceleration, gear and clutch slip

7.2.2 Adversary model

The majority of the existing studies on intrusion detection systems for CAN buses takes into account three different attack types: replay, DoS as well as fuzzing. Such attacks can be straightforwardly detected by verifying that the IDs are or not part of genuine traffic. This is the case for a part of fuzzing attacks where the IDs are random generated as well as for DoS attacks since these are mounted using smaller ID values that have higher priority, in both cases the IDs are not part of legitimate traffic. The replay attacks can be circumvented by ensuring that the periodicity of each ID matches what is anticipated, which is typically not the case for replays.

Additionally, alterations inside the payload can be spotted considering the predicted value of the signal $y'_{\blacklozenge}(k)$ as well as the bias b, a change detection method is applied, which examines the subsequent recurrent sum [137]:

$$S_{\blacklozenge}(k) = \max\big\{0, S_{\blacklozenge}(k-1) + |y_{\blacklozenge}(k) - y'_{\blacklozenge}(k)| - b\big\}, \tag{7.4}$$

where $S(0) = 0$. An intrusion can be spotted by contrasting this cumulative sum with an empirically determined threshold τ. This change detection method, known by the acronym CUSUM, was firstly proposed in [138] and is regularly employed in intrusion detection systems. It is also used in a recent work on intrusion detection for J1939 CAN buses [97], however in the current evaluation we account for the control system level as well as stealthy attacks, which have not previously been considered in the context of in-vehicle networks.

According to [137], there are three types of stealthy attacks that may bypass the detection system based on cumulative sums (CUSUM): surge attacks, bias attacks as well as geometric attacks. Such kind of adversarial manipulations have not been taken into account in prior researches on CAN bus intrusion detection. Consequently, we concentrate particularly on these attacks in this chapter. Since replay as well as DoS attacks can be easily spotted by checking the periodicity of frames (both of these attacks involve many frames, which is uncommon given that CAN IDs are transmitted at regular intervals), the current adversary model accounts for the following types of modification attacks:

1. *fuzzing attacks* – are the modification attacks in which randomly generated signal values are injected in the payload of CAN messages,

2. *surge attacks* – are the modification attacks in which the value of the attack signal is configured to the maximum value or minimum value with the assumption that doing so will cause the maximum damage for the vehicle and yet remain undetected. In this scenario, the attack signal value at step $k+1$ will be $y_{\blacklozenge,\max}$ only if the related sum at the next step is $S_{\blacklozenge}(k+1) \leq \tau$, whereas if not, the attack signal value will remain at $y'_{\blacklozenge}(k) + |\tau + b - S_{\blacklozenge}(k)|$,

3. *bias attacks* – are the modification attacks in which the malicious signal value is

altered by just adding a minor constant $c = \tau/n + b$. i.e., $\widetilde{y}_{\blacklozenge}(k) \leftarrow y_{\blacklozenge}(k) + \tau/n + b$, in order to guarantee that the attack goes undetected for n steps,

4. *geometric attacks* - are the modification attacks in which the malicious signal value is altered by just adding a small drift in the beginning and then the drift grows steadily in the following steps depending on a geometric expansion, i.e., $\widetilde{y}_{\blacklozenge}(k) \leftarrow y_{\blacklozenge}(k) + \beta\alpha^{n-k}$ where α is fixed and $\beta = \frac{(\tau+nb)(\alpha^{-1}-1)}{1-\alpha^{n}}$.

The addressed adversary model and the corresponding countermeasures do not depend on what entry point the adversary exploits. As mentioned previously, the majority of attacks account for an adversary connecting to the bus or remotely compromising a device. Notably, a recent study [139] suggests an ingenious approach in which standard CAN frames are concealed inside CAN-FD frames. Indeed, such kind of attack has not been previously investigated. However, the suggested IDS is intended to run locally on each node, thus regardless of how the attack is carried out, it should be spotted by the IDS when the payload of the frame is examined. Therefore, the proposed IDS should offer resistance to this type of attack as well.

The proposed adversary model supposes that the intruder has a predefined (fixed) success probability of altering the payload inside the frames. In what follows, the justification behind this model is presented. From a networking point of view, it is true that attacks on the CAN bus are frequently carried out by injecting extra (attack) frames into the network, but this gives a slightly inaccurate perception of how in-vehicle controllers work. To put it more specifically, in-vehicle controllers execute cyclic activities that are planned at fixed time intervals, such as 10-100 ms, and consume one CAN frame through the course of each succeeding execution. The frames are typically stored in a buffer for each CAN ID and a new received frame overwrites the previous one. As a result, the controller will only consume the most recent message, even if an adversary injects several attack frames between two genuine ones. Attacks are therefore probabilistic since there are many attack frames competing with legitimate ones on the bus. This is the reason why, instead of injecting messages, a fixed probability of whether a message is corrupted or not is predefined. The above rationale is in line with the reality at the controller level discussed in this work.

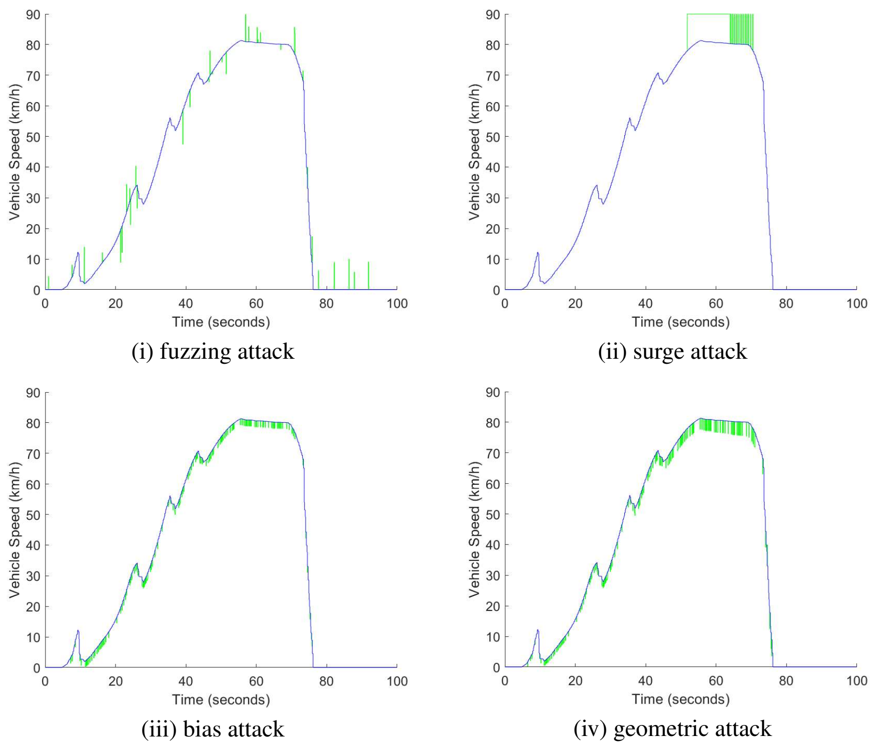
The proposed adversary model can be applied in multiple ways. The first one is the obvious scenario in which a compromised task that runs on the ECU effectively replaces the genuine frames with malicious ones. In addition to this scenario, there are two further aspects at the networking layer that contribute to making the adversary model (which instead of injecting additional messages, replaces genuine frames with compromised ones) more plausible. One is a scenario in which the ECUs are the victims of *bus off* attacks, which leave ECUs passive for a period of time and allows an adversary to inject corrupted messages. The first example of such attacks were demonstrated in [121]. The second is the possibility of messages being altered if they are forwarded by a compromised gateway. Nevertheless, the vehicles are currently evolving from domain oriented architectures, where the ECUs that are responsible for similar features are placed on the same bus to zone oriented architectures in which the ECUs are organized under a zonal controller in accordance with the place where they perform. Even communications that are accountable for the same functionality may need to be routed by gateways under this more recent paradigm, increasing the probability that the frames may be corrupted.

The chapter proceeds with the experimental analysis considering the context of the adversary model mentioned above.

7.3 Experimental results

This section presents the behavior of J1939 signals during adversarial manipulations, the accuracy results in detecting intrusions as well as the computational results on automotive devices, i.e., in-vehicle controllers.

7.3.1 Specific attacks on J1939 signals



(i) fuzzing attack (ii) surge attack

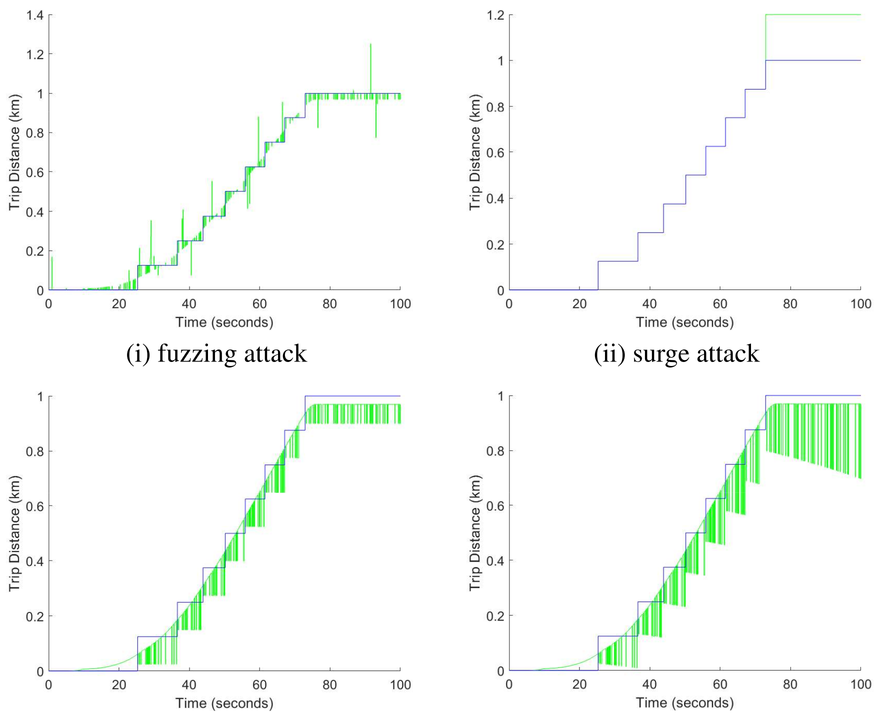
(iii) bias attack (iv) geometric attack

Figure 7.10: The variation of vehicle speed signal under various attack scenarios

For current evaluation the focus is concentrated on J1939 specific signals for which the J1939-71 standard [81] provides an easy way to identify them. The evaluation specifically takes into account the following four parameters:

1. vehicle speed measured in (km/h),
2. trip distance measured in (km),
3. engine speed measured in (rpm),
4. engine torque measured in (Nm).

For a better understanding of the behavior of the attacks and the effects they cause, a graphic representation with the J1939 specific signal fluctuations during various attacks take place, is shown in Figures 7.10, 7.11, 7.12 and 7.13.

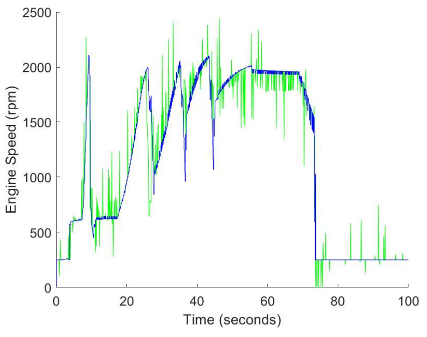


(i) fuzzing attack (ii) surge attack

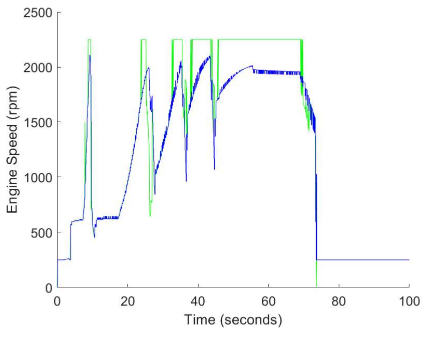
(iii) bias attack (iv) geometric attack

Figure 7.11: The variation of trip distance signal under various attack scenarios

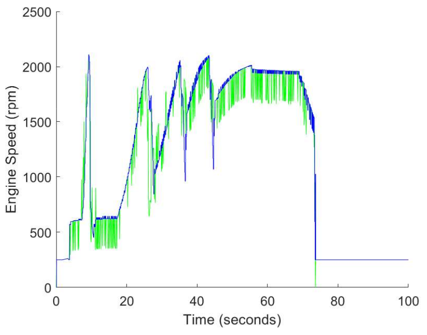
The fluctuation of the vehicle speed signal is depicted in Figure 7.10. The alterations caused by the adversary are highlighted with green, while the valid signal appears with blue. The fuzzing attacks, which take the form of spikes surrounding the genuine signal, are depicted in Figure 7.10 (i). Figure 7.10 (ii) shows that during a surge attack, the



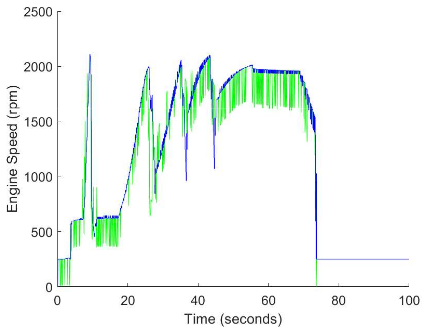
(i) fuzzing attack



(ii) surge attack



(iii) bias attack



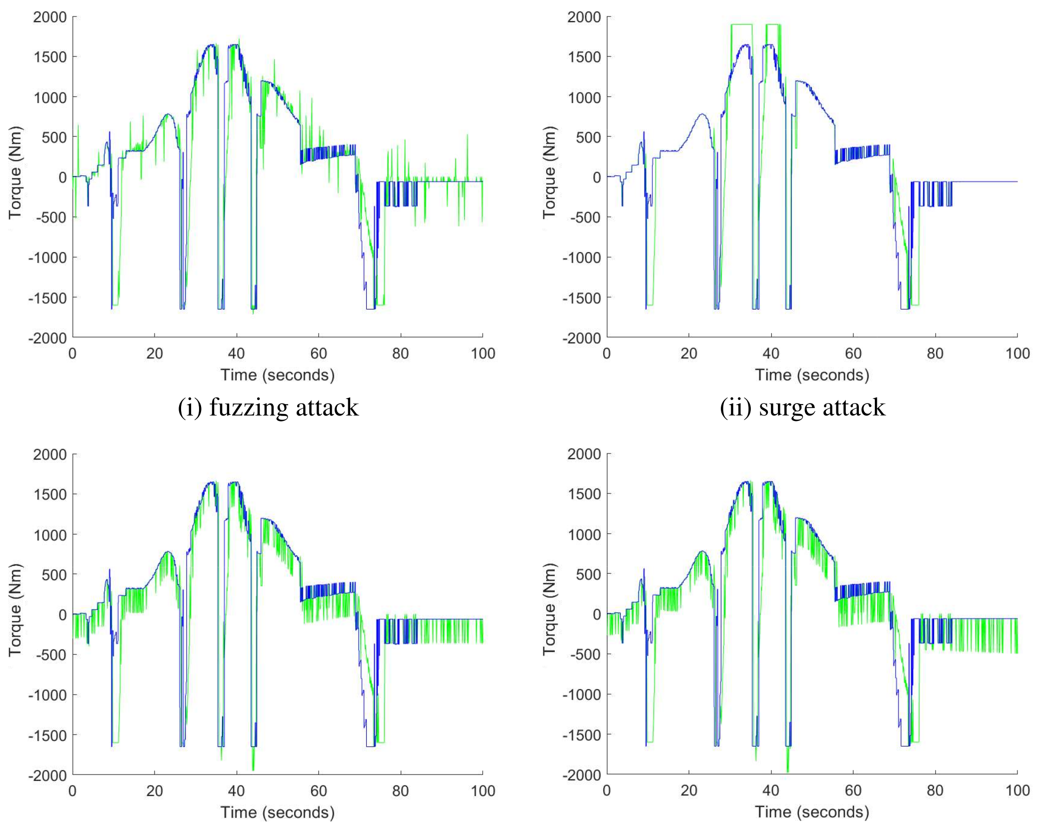
(iv) geometric attack

Figure 7.12: The variation of engine speed signal under various attack scenarios

falsified signal is kept at 90 km/h, but this only occurs when the genuine signal gets close to the attack value. When bias attacks from Figure 7.10 (iii) and geometric attacks from 7.10 (iv) take place, the valid signals are only slightly deviated.

Bias attacks and geometric attacks differ in that the distance between the legitimate signal and the attack signal stays constant for the former while for the latter it gradually increases. The trip distance signal is incremented for every 125 m traveled and due to this, exhibits a different waveform in comparison with other signals, as can be observed in Figure 7.11. As seen in Figure 7.11 (ii), the manipulated signal reaches a top level of 1.2 km when surge attacks take place. Figures 7.11 (iii) and (iv) show the behavior for the bias and geometric attacks, which is similar to the behavior for the preceding signal. As depicted in Figure 7.12, the genuine engine speed signal exhibits a larger range of values, which is 0–2050 rpm. In this scenario, when a surge attack takes place, the altered signal is equal with 2250 rpm, as shown Figure 7.12 (ii). When the bias and geometric attacks occurs, as depicted in Figures 7.12 (iii) and (iv), the manipulated signals exhibit higher deviations from the genuine signals in comparison with the ones accounted for the vehicle speed signal.

The attacks mounted on the torque signal are presented in Figure 7.13. They exhibit a behavior that is slightly similar to the one recorded for the engine speed except that the signal also has negative values.



(i) fuzzing attack

(ii) surge attack

(iii) bias attack

(iv) geometric attack

Figure 7.13: The variation of torque signal under various attack scenarios

7.3.2 Results on detecting intrusions

Before providing the detection results, an overview of how the adversary behaves in the current setup is presented. The adversary is configured to operate with a specified attack probability p_{att} for each parameter of the payload. This corresponds to the common adversary model in which the frames that are manipulated by an adversary are injected into genuine traffic. This attack probability is used in case of fuzzing, bias, as well as geometric attacks, whereas the surge attack only occurs if the prerequisites outlined in the adversary model section are satisfied. The attacks that produce signal values outside of the expected range, e.g, negative signal values while the expected ones are positive, will not be mounted because the range checks can spot them right away. The IDS (change detection mechanism) operates inside the Simulink environment and exchanges data with the CANoe environment interacting with it in synchronized mode.

Table 7.2: Detection results for engine speed based on machine leaning algorithms

Algorithm	**Attack** type	**FPR** (%)	**FNR** (%)	**Accuracy** (%)	**Precision** (%)
k-NN	fuzzing	16.20	74.87	69.50	33.33
k-NN	surge	1.18	0.00	99.38	98.68
k-NN	bias	1.06	98.54	82.25	22.22
k-NN	geom	0.15	98.26	83.00	50.00
DTC	fuzzing	28.26	37.95	69.38	41.44
DTC	surge	0.00	0.00	100.00	100.00
DTC	bias	0.45	100.00	82.50	0.00
DTC	geometric	9.19	99.26	75.50	1.61
RFC	fuzzing	12.73	55.38	76.88	53.05
RFC	surge	0.00	0.00	100.00	100.00
RFC	bias	0.60	100.00	82.38	0.00
RFC	geometric	0.15	99.26	83.00	50.00

The results show that the machine learning algorithms are incapable of defending against modification attacks. These machine learning based IDSs are deployed using the Statistics and Machine Learning Toolbox made available by MATLAB and the Python scikit-learn package `https://scikit-learn.org/stable/`. For the C code generation the sklearn-porter library `https://github.com/nok/sklearn-porter` is employed. The generated code is then ported on embedded boards to determine the time needed for the algorithms in order to classify one frame. The classification models are trained using parts of the logged CAN traffic.

The CAN trace is split in two portions: 20% is for training purposes while the rest of 80% is employed for testing. The detection results from Table 7.2 are obtained using the following common machine learning classifiers:

- k-nearest neighbors – k-NN,
- decision trees classifier – DTC,
- random forest classifier – RFC.

Only frames containing the targeted signal were subject to the attack. As can be observed, the IDS based on these classifiers has significant shortcomings in terms of detecting fuzzing, bias, and geometric attacks. The only attacks against which the IDS is effective are the surge attacks. Bias and geometric attacks exhibit a false negative rate of over 98%, hence they go mostly undetected. In case of fuzzing attacks the FNR is lower, ranging from 37% to 74%, but it is still inadequate. Table cells with poorer precision are highlighted in gray.

Table 7.3: Adversary and IDS parameters used for different types of attacks on Vehicle Speed, Trip Distance, Engine Speed and Torque

Adversary / IDS parameters					
No.	**J1939 signal**	**Att. type**	**Att. prob.**	**Bias Adv. / IDS**	**Threshold Adv. / IDS**
1.	Vehicle Speed (km/h)	fuzzing	25%	- / 0.5	- / 2
2.		surge	-	5 / 0.5	10 / 2
3.		bias	25%	5 / 0.5	10 / 2
4.		geometric	25%	5 / 0.5	10 / 2
5.	Trip Distance (km)	fuzzing	25%	- / 0.05	- / 0.075
6.		surge	-	0.2 / 0.05	0.25 / 0.075
7.		bias	25%	0.2 / 0.05	0.25 / 0.075
8.		geometric	25%	0.2 / 0.05	0.25 / 0.075
9.	Engine Speed (rpm)	fuzzing	25%	- / 100	- / 150
10.		surge	-	400 / 100	600 / 150
11.		bias	25%	400 / 100	600 / 150
12.		geometric	25%	400 / 100	600 / 150
13.	Torque (Nm)	fuzzing	25%	- / 200	- / 250
14.		surge	-	500 / 200	1000 / 250
15.		bias	25%	500 / 200	1000 / 250
16.		geometric	25%	500 / 200	1000 / 250

On the other hand, the change detection method works remarkably well. The rationale behind selecting the thresholds values which are shown in Table 7.3 as well as the accuracy results on detecting intrusions presented in Tables 7.4 and 7.5, are discussed next. The rows with detection results from Tables 7.4 and 7.5 correspond to the rows with attack scenarios and parameters accounted in Table 7.3, they can easily identified with the help of the first column indicating the assessment number. The values for bias and threshold were empirically established. Both values were determined for the IDS and for the adversary as well. If both the adversary and the IDS use the same threshold values, only minor modifications from the genuine parameters take place. In this case, the impact is minimal and the adversarial manipulations remains undetected by the IDS. The IDS threshold, for instance, is 2 km/h, while the threshold for engine speed is 150 rpm. Regardless of whether the adversary produces these deviations in order to confuse the driver, they will likely have no impact. Consequently, it is expected that the adversary will try to produce greater deviations that can more effectively deceive the driver, e.g, increasing or decreasing the reported speed by 10 km/h. This clarifies why the thresholds from Table 7.3 were selected.

Table 7.4: Results in detecting intrusions on Vehicle Speed, Trip Distance, Engine Speed and Torque – short 100 s simulation (1000 CAN frames)

Detection results								
No.	**TN (frames)**	**TP (frames)**	**FP (frames)**	**FN (frames)**	**FPR (%)**	**FNR (%)**	**Accuracy (%)**	**Precision (%)**
1.	759	233	0	8	0.00	3.32	99.20	100.00
2.	734	266	0	0	0.00	0.00	100.00	100.00
3.	842	158	0	0	0.00	0.00	100.00	100.00
4.	838	162	0	0	0.00	0.00	100.00	100.00
5.	759	229	0	12	0.00	4.98	98.80	100.00
6.	614	386	0	0	0.00	0.00	100.00	100.00
7.	842	158	0	0	0.00	0.00	100.00	100.00
8.	847	153	0	0	0.00	0.00	100.00	100.00
9.	753	196	6	45	0.79	18.67	94.90	97.03
10.	636	357	7	0	1.09	0.00	99.30	98.08
11.	820	172	6	2	0.73	1.15	99.20	96.63
12.	820	174	6	0	0.73	0.00	99.40	96.67
13.	752	192	7	49	0.92	20.33	94.40	96.48
14.	698	277	12	13	1.69	4.48	97.50	95.85
15.	752	230	7	11	0.92	4.56	98.20	97.05
16.	752	239	7	2	0.92	0.83	99.10	97.15

If the adversary thresholds values are larger than the IDS threshold values then the attack can produce a major impact, however, this will be spotted by the IDS. The detection results obtained for vehicle speed and trip distance signals exhibits an excellent accuracy and precision that are around 100%. This is the case since the both signals have been accurately predicted. On the other hand, the detection accuracy and precision obtained for the engine speed as well as torque signals, are lower, at 94%–99% and 95%–98%, respectively. The FPR and FNR have increased, but at 0%-1% and 0%-20%, respectively, these still appear reasonable. This is because the estimation accuracy was not sufficiently good as the last two genuine signals were predicted using the previously specified lookup tables. Better predictions can be made using the vehicle mechanical data, which are outside the scope for the current investigation.

The prior results were obtained using a simulation that lasts 100 seconds and the simulation step has 0.1 second, yielding a total of 1000 CAN frames. It is true that other studies based on CAN bus data recorded inside vehicles employ traces that last for at least an hour. This strategy to use long traces is not always optimal since the complexity of detecting attacks is determined by the variability of the CAN traffic rather than the length of the trace. The small trace (1000 CAN frames) that is used in the current evaluation contains CAN traffic that exhibits considerable variations of the logged J1939 specific signals. This trace (1000 CAN frames) contains CAN traffic that exhibits considerable variations for the logged J1939 specific signals whereas a longer trace of several hours in which the CAN traffic does not show noticeable variations is not always more convincing.

Table 7.5: Results in detecting intrusions on Vehicle Speed, Trip Distance, Engine Speed and Torque – long 1 hour simulation (36000 CAN frames)

				Detection results				
No.	**TN**	**TP**	**FP**	**FN**	**FPR**	**FNR**	**Accuracy**	**Precision**
	(frames)	**(frames)**	**(frames)**	**(frames)**	**(%)**	**(%)**	**(%)**	**(%)**
1.	26988	8676	0	336	0.00	3.73	99.07	100.00
2.	2036	33964	0	0	0.00	0.00	100.00	100.00
3.	27202	8798	0	0	0.00	0.00	100.00	100.00
4.	27192	8808	0	0	0.00	0.00	100.00	100.00
5.	26988	8974	0	38	0.00	0.42	99.89	100.00
6.	35221	779	0	0	0.00	0.00	100.00	100.00
7.	27071	8929	0	0	0.00	0.00	100.00	100.00
8.	27076	8422	0	502	0.00	5.63	98.61	100.00
9.	26851	7481	137	1531	0.51	16.99	95.37	98.20
10.	25145	10758	97	0	0.38	0.00	99.73	99.11
11.	27021	8803	137	39	0.50	0.44	99.51	98.47
12.	27021	8798	137	44	0.50	0.50	99.50	98.47
13.	26805	7023	183	1989	0.68	22.07	93.97	97.46
14.	34352	1450	179	19	0.52	1.29	99.45	89.01
15.	26805	8828	183	184	0.68	2.04	98.98	97.97
16.	26805	8964	183	48	0.68	0.53	99.36	98.00

In what follows, the detection mechanism is evaluated based on a trace that lasts an hour to demonstrate this. In this scope, an even more complex CAN traffic including plenty of specific activities, such as gear shifts, braking suddenly, speed fluctuations, etc., was generated. Table 7.5 shows the detection results. With a few exceptions, the number of false positives and negatives substantially decreased when compared to the short scenario (only 1000 frames). In other words, the false positive rate is now below 1%, and the false negatives rate is between 0% and 5%, except for fuzzying attacks where the FNR ranges from 0% to 22%. A closer investigation revealed the reason for the decrease in false positives: roughly all of them are brought on by gear shifts which produce in turn abruptly variations on engine speed and torque signals that are difficult to predict with the lookup tables. Over the course of an hour, the long trace accounts around 30 gear shifts, however the short trace exhibits 10 gear shifts in only 100 seconds. This indicates that the sudden variations in engine speed or torque that are recorded when a gear shift take place might be disregarded in order to prevent false alarms.

Since unexpected gear shifts can happen for a variety of seemingly unimportant reasons, it is challenging to look deeper into this matter. To reduce the vehicle speed or to boost the torque when dealing with a slope, for instance, a driver could downshift gears rather than press the brake pedal. This is possible in both circumstances, whether there is a manual gearbox or an ECU specifically designed to handle these actions. All of these behaviors, which in fact are not attacks, could result in abrupt variations of the engine speed and torque signals.

By examining some relevant papers on CAN IDS, we can see that the reported FPR is 1.60% in [59], 4.68% in [140], or even 6.45% in [141]. The study from [18] relies on cryptographic security and reports an FPR of 0%. On the other hand, the FNR is 2.80% in [59], 2.40% in [140] as well as 3.90% in [141]. Once more, the authors from [18] report a 0.01% FNR but achieves this with cryptographic security. Consequently, the only option if higher FPR and FNR are required would be to rely on cryptography and embed security elements inside CAN frames. The AUTOSAR SecOC [129] recommends to pack inside each CAN frame 32 bits of security elements, i.e., an authentication tag as well as a freshness parameter. In accordance with security profiles 1–3 [129], the authentication tag in this instance is 24–28 bits, and the probability to mount a succesfull attack is 2^{-24} to 2^{-28}. However, keep in mind that due to the 0–8 bits small size of the freshness parameter (depending on the security profile), after just 256 messages this parameter is repeated. As a result, the probability of successful replay attack remains at 2^{-8}, which is not negligible.

The previously mentioned attacks have an immediate effect on safety because many ADAS (Advanced Driver Assistance Systems) systems rely on signals like engine speed, torque, and vehicle speed. These signals can determine the car to accelerate or decelerate unexpectedly, which might result in collisions. When predicting the time until a collision occurs, keep in mind that a simple 5 km/h deviation from the real vehicle speed translates into 1 m/s, which could have severe repercussions.

7.3.3 Runtime performance

The computational performance evaluation of DT and RFC classifiers as well as of change detection mechanism was conducted on four automotive-graded platforms. These platforms, named S12XF, SAM V71, TC277, and TC297, are produced by reputable manufacturers: NXP, Microchip, and for the two remaining ones Infineon. The first one, which serves as a benchmark for the low-end device sector, offers 32 KB of RAM as well as 512 KB of Flash memory and operates at a maximum frequency of 50 MHz. On the other hand, the other three platforms are candidates for high-end device sector and operates on 32 bits. The Microchip SAM V71 Xplained Ultra platform is enabled by an ATSAMV71Q21 microcontroller that complies with the ARM Cortex M7 specification and provides a top operating frequency of 300 MHz and is equipped with 2 MB of Flash and RAM.

Both the TC277 and TC297 controllers from Infineon have been designed based on the Aurix TriCore architecture, however they exhibits different top operating frequencies. The first can operate at a maximum frequency of 200 MHz, while the second can be

Table 7.6: Main features of the used embedded development boards

Feature	**TC277**	**TC297**	**SAM V71**
Architecture	TriCore	TriCore	ARM
CPU cores	3	3	1
Max. frequency	200MHz	300MHz	300MHz
FLASH	4MB	8MB	2MB
RAM	472KB	728KB	2MB
EEPROM	384KB	768KB	256KB

clocked with a frequency at up to 300 MHz. All these aforementioned features are summarized in Table 7.6. Figure 7.14 shows the experimental setup used for the evaluation of computational performance. Every board was configured to operate at its top supported frequency.

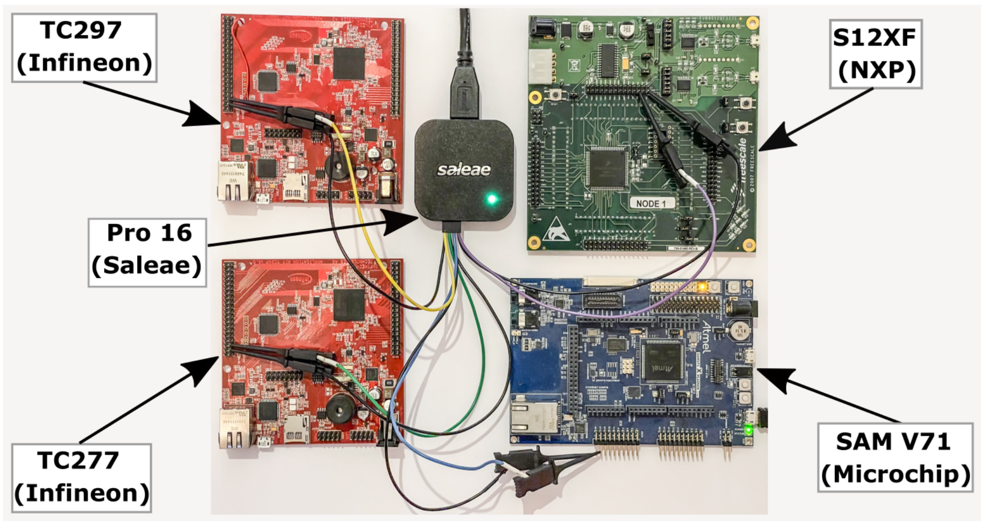


Figure 7.14: Experimental setup used for run-time assessment

Both DTC and RFC, which were employed as detection algorithms, were programmed to run with the default settings, and for RFC there are 15 estimators configured. For the k-NN implementation, a lot of memory is required to store the neighbors data for genuine and malicious frames. As a result, this was not ported on the embedded platforms. The runtime results are listed in Table 7.7. The results obtained for the engine speed signal are the only ones reported because the runtimes for the other signals are similar. The classifier that was used and the type of attacks are listed in columns two and three, respectively. The final four columns provide the runtime measured for each board.

Given that the RFC uses more estimators, it should come as no surprise that the RFC classifier operates much slower than the DTC. The runtime for both machine learning classifiers on the low-end device, i.e S12, ranges from 60.74 to 715 μs. On the other hand, the measured runtime required to execute the less complex change detection technique is only 39 μs microseconds. The time needed for the execution of machine learning classifiers varies from 0.74 to 18.89 μs on high-end platforms while for the change detection technique varies from 0.13 to 1.32 μs. This shows that in addition to the higher accuracy results on detecting intrusions demonstrated in the preceding section, the change detection mechanism is executed 2-65 times faster. Furthermore, compared to machine learning based algorithms, which have a footprint of 10–30 KB, the change detection mechanism requires minimal memory requirements – it just needs several lines of code.

Table 7.7: Computational results of IDS algorithms on automotive graded controllers

J1939 sig.	**Algorithm**	**Attack**	**Execution time on target (μs)**			
			S12XF	**SAM V71**	**TC277**	**TC297**
Engine Speed	DTC	fuzzing	199.12	18.89	2.34	1.86
		surge	60.74	5.49	0.91	0.74
		bias	130.23	12.48	1.64	1.28
		geometric	153.83	15.07	1.94	1.51
	RFC	fuzzing	575.20	16.92	13.12	8.55
		surge	355.65	7.69	7.41	5.35
		bias	620.38	15.44	8.91	6.41
		geometric	715.00	16.98	9.15	6.56
	Change detection	**all**	**39.10**	**1.32**	**0.34**	**0.13**

7.4 Concluding remarks

The analysis from this chapter demonstrates that, despite growing efforts to address CAN bus intrusions using machine learning techniques, such algorithms are unable to detect subtle alterations inside the payload. The deficiencies stem from two obvious factors: the restricted training time, which means that the training dataset cannot cover all achievable car states, and the minor alterations that adversaries might introduce to make the attack stealthy. On the other hand, by employing specific change detection mechanism, the detection results are better and only a significantly low computational cost is induced, as proven by the experiments. Although redundancy is necessary for the deployment of such mechanisms, we have demonstrated that it is feasible in the J1939 model, which was used for our tests. We therefore hope that the work from this chapter open the road in this direction because there are few or no studies that cover such fine grain features.

Chapter 8

Conclusion

This thesis investigates various intrusion detection systems for CAN buses with a focus on the SAE J1939 heavy-duty vehicle deployments. The approaches utilized for the deployment of these IDSs range from the use of machine learning algorithms to hindering adversaries by the use of symmetric cryptography or making a fine grained analysis at the control systems level. The majority of the results from this thesis are based on real-world CAN traffic that was collected by the author either from a commercial vehicle (a modern agricultural machinery) or from passenger cars (a sedan or SUV). Part of the results are based on traces generated directly from the CANoe, an industry-standard tool that is widely used in the implementation of in-vehicle networks. To test and demonstrate that the detection mechanisms are suitable for real-world deployment inside vehicles, several measurements on their runtime are performed on automotive devices, i.e., in-vehicle controllers. Moreover, the performance results in detecting intrusions obtained using the proposed solutions are compared with the ones reported by the related works on intrusion detection for CAN buses in order to highlight the advantages offered by them. A brief overview of each chapter follows, also highlighting some of the significant findings.

Chapter 3 gives an overview on CAN protocol in the context of the SAE J1939 compliant CAN buses. Some details on the CAN physical layer, as well as the structure of the data frame, are presented. The chapter continues with a decription of SAE J1939 specifics such as: parameter group numbers, transport protocols, frame identifier breakdown as well as the address claiming procedure. All of these are discussed in order to set room for the deployment and evaluation of specific security solutions tailored to SAE J1939 CAN buses that are presented in the next chapters.

Chapter 4 explores the use of neural networks as candidates for intrusion detection in Controller Area Networks. The first section of this chapter starts with a presentation of the scenarios that can be exploited by an adversary to gain access to the CAN bus via the OBD port and injects adversarial CAN frames. The same section details the types of attacks that are subject for the evaluation of the IDS. The following section describes the development environments employed for the implementation of neural network based intrusion detection. Two different deployments are done, one using the Neural Network Toolbox made available by the MATLAB platform while the other is based on a C++ implementation. The first approach is preferred since the neural network toolkit offered by MATLAB is well-known for its performance, while the second is used since the C++ code can be easily ported on automotive-graded controllers. Nevertheless, a verification is done to make sure that identical detection results are produced using both deployments, and indeed this is the case. Then, this section briefly discusses the stopping conditions (for the training stage), the neural network architecture as well as the features extracted from the CAN traffic, which serve as inputs for the neural network. The experimental results in terms of detection and runtime performance for the proposed IDS are detailed in the next section from this chapter. The experiments are performed on a J1939 compliant CAN traffic, recorded using a CANoe simulation and on a public CAN dataset. For each type of attack (replay or injections with random data inside the payload), several scenarios are tested depending on how the CAN traffic is split between training, validation and testing as well as on how the attacks are performed, either on a single ID or on full trace (all IDs). Moreover, in order to emphasize the trade-off between the accuracy of the detection results and the runtime performance, the evaluation is carried out using three different sizes for the neural network. The runtime performance of the detection algorithm is measured on three automotive graded controllers, one regarded as candidate for the low-end device group and the other two for the high-end device group. The proposed IDS has a poorer accuracy in detecting replay attacks since the content of the injected frames is the same to the one of the genuine frames (the arrival time of the CAN frames is the sole way to identify such attacks). Overall, the experiments show that despite the good results in detecting intrusions, the usage of such a solution for the real-time filtering of the CAN traffic is debatable at least on microcontrollers from the low-end device group, since the time required to classify a CAN frame is in order of dozens milliseconds. This can be regarded as an important aspect since most of the related papers do not present such results, or use a PC based environment with powerful resources, which does not reflect the reality at microcontroller level.

A framework that enables the integration of various CAN bus attacks as well as intrusion detection systems inside a CANoe based simulation and provides a realistic testbed for them, is presented in Chapter 5. The first section of this chapter begins with a description of the data collection procedure used by the author for extracting the CAN traffic from the two passenger cars. Then, this section presents the setup employed for the data extraction inside vehicles as well as the format of the recorded CAN messages. The CAN network structure from the CANoe simulation and how the collected CAN traffic is used inside it, are detailed at the end of this section. The second section introduces various types of adversarial manipulations that were integrated into the CANoe simulation. These attacks include replays, injections with randomly generated data inside the payload, injections with scalar addition or multiplication of the payload content as well as arbitrary injections. Then, this section proceeds with a presentation of how the designed graphical interface, which permits to configure various adversary parameters, can be used. In the following section an overview of the *Statistics and Machine Learning Toolbox* offered by MATLAB platform as well as of the k-NN algorithm, a candidate for IDS, are presented. This section ends with a description of the input sample for the k-NN algorithm, which contains the following features extracted from the CAN frames: the time interval elapsed between consecutive arrivals of CAN messages that share the same identifier as well as the data field content. Last but not least, the experimental section discussed the detection results obtained for several scenarios. The experiments are conducted to cover all the previously defined types of attacks. The tested scenarios depend on the attack delay, on what the adversary targets (one ID or full CAN traffic), on how many neighbors and which distance metric are used by the k-NN algorithm as well as on the scalar values utilized for the addition and multiplication of payload bytes. As far as the author is aware, this is the first framework that allows the simulation of both the CAN bus attacks and the IDS inside a unified environment. At the same time, using a simulated environment is safer because it prevents any potential danger to drivers or cars.

In Chapter 6, a targeted intrusion detection and prevention system for securing the SAE J1939 heavy-duty CAN buses is evaluated. The first section summarizes how the J1939 compliant CAN traffic is collected by the author through the J1939 specific diagnostic port from an agricultural machinery (a tractor). Following that, this section continues with an inspection of the J1939 compliant features that are present inside the collected traffic. The recorded traffic contains 51 frame identifiers and it is transmitted by three different ECUs that are determined by investigating the unique source addresses that are embed inside the IDs. According to J1939 standardization, the function of each ECU from the network becomes apparent as follows: engine control module, body control module and transmission control module, respectively. A quantitative analysis focusing on the periodicity accounted for each CAN ID from the recorded CAN messages, is discussed at the end of this section. The next section introduces the adversary model which includes both replays and modification attacks. Nevertheless, the modification attacks follow a different approach from the one used in Chapters 4 and 5 in that they are mounted at parameter level, not on the complete payload. This section then presents the two-stage IDS that was designed to protect J1939 CAN buses. The first stage relies on the *encrypted addresses*, i.e., the source as well as the destination addresses from each frame ID are encrypted using AES. The rationale behind embedding security elements into CAN identifiers is as follows. According to J1939 standardization, the payload of the J1939 CAN frames is fully allocated with specific J1939 parameters, which are assigned to a parameter group. Security elements are therefore packed in the IDs rather than the payload because doing otherwise would have been in contrast with J1939 specifications. This stage is complemented by the second one, which is based on the encrypted payload. This enables the detection even when single bits are tampered, due to the avalanche effect resulted from the decryption of the payload and subsequent plausibility checks applied for each J1939 parameter carried by the frame. Subsequently, this section discusses how these encrypted addresses will be generated and stored using an ordered binary tree as well as circular lists. Since these addresses must be generated cyclically. The section ended with a formalisation on the trade-off between the periodicity of address tree generation and resources (computational power and storage capacity). Last but not least, the experimental section gives a presentation of runtimes measured on automotive development boards for the symmetric encryption operations as well as for the generation of the address tree. Then, the section gives an overview of the active defense mechanism, which acts like a prevention system. This mechanism permits the decoding of the CAN frame content (ID and payload) before the ACK slot becomes overwritten by receivers in case of a correct reception. Moreover, if the decoded frame is classified as an intrusion, then the destroying procedure of it with error flag is triggered. To test and prove the frame destruction capabilities of the prevention mechanism a laboratory setup is created. Lastly, this section shows the detection results obtained for attacks mounted on different J1939 parameters and provides an analogy between the security level accounted for the proposed solution and one that meets AUTOSAR SecOC requirements [129]. Overall, the proposed solution, which was specifically designed to meet J1939 specifications, proved to be a cost-effective method for both real-time detection and the elimination of attack frames. Moreover, to the best of author's knowledge this is the first intrusion detection and prevention system tailored for SAE J1939 heavy-duty vehicle buses. The innovative approach for decoding the CAN frame using ICU is another significant development of this chapter.

Chapter 7 presents an intrusion detection system at control system level tailored to meet J1939 specifications. An overview on the connection between Simulink and CANoe and their combined operation modes is presented in the first section from this chapter. The next section proceeds with a brief description of the J1939 CANoe Simulation, the employed Simulink models as well as the adversary model. The Simulink models are integrated into the CANoe simulation and allow the prediction of the following signals: vehicle speed, trip distance, engine speed and torque. The adversary model accounts for specific attacks mounted at the control system level such as: surge attacks, bias attacks and geometric attack, which have not been considered by others works on CAN bus intrusion detection until now. Beside these three attack types, also a more common type of attack is taken into consideration, i.e., fuzzing attacks. The first part of the experimental section points out the behavior of the J1939 parameters when such attacks are mounted. Then, this section presents the performance results in detecting intrusion accounted for both the machine learning based approaches and change detection mechanisms. A significant finding from this section shows that, in spite of rising efforts to deploy machine learning approaches as candidates for IDS, such algorithms are unable to spot minor changes in the data field. On the other hand, as demonstrated by the experiments, the change detection mechanism seems to be more effective in detecting such attacks. For each J1939 signal, several attack scenarios are tested with regard to short simulations (1000 CAN frames) and long simulations (1 hour). For each signal, regardless of the attack type, the same bias and threshold values are maintained. This was done to cover the more realistic scenarios. The last part of this section discussed the runtime performance of the proposed IDS algorithms. The obtained results clearly show that the runtime of the change detection mechanisms is 2-65 times smaller than the one required for the execution of the machine learning algorithms. Moreover, not only from the perspective of computational efficiency does the change detection mechanism performs better, but also from the storage capacity point of view – since it requires just several lines of codes, while the machine learning based algorithms have a footprint of 10-30 KB.

All in all, this thesis investigates various security solutions that target SAE J1939 heavy-duty vehicle CAN buses, some of which could also be applied to regular CAN deployments, e.g., from passenger cars. Their practical application has been tested in various laboratory setups using equipment designed specifically for the automotive industry (microcontrollers, VN1640, CAN cables, etc.). As a major result, this thesis accounts for a realistic platform for testing in-vehicle CAN bus attacks and IDS as well, an intrusion detection and prevention systems tailored to meet J1939 specifications, a novel method to interpret the content of the CAN frames (ID and payload) before their correct reception as well as an IDS designed at control system level. The results of this thesis were published in relevant journals and conferences in the field of security, automotive, and industry applications. Overall, the results indicate that the intrusion detection systems are essential in boosting the security of in-vehicle networks and such mechanism can be efficiently deployed.

Nevertheless, the research carried out in this thesis have produced certain results that can serve as a starting point for additional investigations in this direction. For the proposed machine learning based deployments, future work could involve the usage of more powerful cores to enhance the computational efficiency which is crucial for real-time CAN traffic filtering. On the other hand, the solution which relies on cryptography can make use of cryptographic hardware accelerators from the in-vehicle controllers as this would benefit to speed up the security operations. Another future direction, which can be applied for all proposals, is the extension of the assessments on CAN-FD networks, which recently have been adopted for J1939 as well [91]. This extension of CAN could lead to an increase of the computational demands when considering solutions based on machine learning because of the larger amount of data packed inside CAN-FD frames, but it may also pave the way for a better integration of security elements inside the payload.

List of Figures

List of Tables

Bibliography

[1] "Truck registrations in the U.S. from 2000 to 2021," https://www.statista.com/statistics/857374/truck-registrations-in-the-us/, 2023, accessed: 2023-08-10.

[2] T. Ceccarelli, A. Chauhan, G. Rambaldi, I. Kumar, C. Cappello, S. Janssen, and M. McCampbell, "Leveraging automation and digitalization for precision agriculture: Evidence from the case studies," 2022.

[3] "Global agriculture equipment unit sales from 2019 to 2029," https://www.statista.com/statistics/1227934/global-agriculture-equipment-unit-sales/?fbclid=IwAR1ij5JIbNmem-eAPVFtj4l5B3G__CTLfbWH1DfFRCcxYVck1dNKXUxsPY8, 2023, accessed: 2023-08-12.

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

[5] F. Hartwich and R. Bosch, "Introducing CAN XL into CAN Networks," *future*, vol. 11898, p. 1, 2015.

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

[7] 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.

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

[9] I. Foster, A. Prudhomme, K. Koscher, and S. Savage, "Fast and vulnerable: A story of telematic failures," in *9th USENIX Workshop on Offensive Technologies*

(WOOT 15). Washington, D.C.: USENIX Association, Aug. 2015. [Online]. Available: https://www.usenix.org/conference/woot15/workshop-program/presentation/foster

[10] S. Nie, L. Liu, and Y. Du, “Free-Fall: Hacking Tesla from Wireless to CAN Bus,” *Briefing, Black Hat USA*, vol. 25, pp. 1–16, 2017.

[11] S. Nie, L. Liu, Y. Du, and W. Zhang, “Over-the-air: How we remotely compromised the gateway, BCM, and autopilot ECUs of Tesla cars,” in *Black Hat USA*, August 2018, pp. 1–19.

[12] H. Wen, Q. A. Chen, and Z. Lin, “Plug-N-Pwned: Comprehensive Vulnerability Analysis of OBD-II Dongles as A New Over-the-Air Attack Surface in Automotive IoT,” in *29th USENIX Security Symposium (USENIX Security 20)*, 2020, pp. 949–965.

[13] S. Woo, H. J. Jo, and D. H. Lee, “A Practical Wireless Attack on the Connected Car and Security Protocol for In-Vehicle CAN,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, no. 2, pp. 993–1006, 2015.

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

[15] “J1939 - Serial Control and Communications Heavy-Duty Vehicle Network,” SAE International, Standard, June. 2023.

[16] Y. Burakova, B. Hass, L. Millar, and A. Weimerskirch, “Truck Hacking: An Experimental Analysis of the SAE J1939 Standard,” in *10th USENIX Workshop on Offensive Technologies (WOOT 16)*, 2016.

[17] S. Mukherjee, H. Shirazi, I. Ray, J. Daily, and R. Gamble, “Practical DoS Attacks on Embedded Networks in Commercial Vehicles,” in *Information Systems Security*, I. Ray, M. S. Gaur, M. Conti, D. Sanghi, and V. Kamakoti, Eds. Cham: Springer International Publishing, 2016, pp. 23–42.

[18] C. Jichici, B. Groza, R. Ragobete, P.-S. Murvay, and T. Andreica, “Effective Intrusion Detection and Prevention for the Commercial Vehicle SAE J1939 CAN Bus,” *IEEE Transactions on Intelligent Transportation Systems*, 2022.

[19] C. Jichici, B. Groza, and P.-S. Murvay, “Integrating Adversary Models and Intrusion Detection Systems for In-vehicle Networks in CANoe,” in *International Conference on Information Technology and Communications Security*. Springer, 2020, pp. 241–256.

[20] T. Andreica, C.-D. Curiac, C. Jichici, and B. Groza, “Android Head Units vs. In-Vehicle ECUs: Performance Assessment for Deploying In-Vehicle Intrusion

Detection Systems for the CAN Bus," *IEEE Access*, vol. 10, pp. 95 161–95 178, 2022.

[21] L. Popa, B. Groza, C. Jichici, and P.-S. Murvay, "ECUPrint—Physical Fingerprinting Electronic Control Units on CAN Buses Inside Cars and SAE J1939 Compliant Vehicles," *IEEE Transactions on Information Forensics and Security*, vol. 17, pp. 1185–1200, 2022.

[22] C. Jichici, B. Groza, and P.-S. Murvay, "Examining the Use of Neural Networks for Intrusion Detection in Controller Area Networks," in *International Conference on Security for Information Technology and Communications.* Springer, 2019, pp. 109–125.

[23] C. Jichici, A. Berdich, A. Musuroi, and B. Groza, "Control System Level Intrusion Detection on J1939 Heavy-Duty Vehicle Buses," *IEEE Transactions on Industrial Informatics*, pp. 1–13, 2023.

[24] B. Groza, P.-S. Murvay, L. Popa, and C. Jichici, "CAN-SQUARE – Decimeter Level Localization of Electronic Control Units on CAN Buses," in *Computer Security–ESORICS 2021: 26th European Symposium on Research in Computer Security, Darmstadt, Germany, October 4–8, 2021, Proceedings, Part I 26.* Springer, 2021, pp. 668–690.

[25] "ISO/SAE 21434:2021 Road vehicles — Cybersecurity engineering," International Organization for Standardization, Standard, 1st edition, Aug 2021.

[26] L. Popa, C. Jichici, T. Andreica, P.-S. Murvay, and B. Groza, "Impact of Wiring Characteristics on Voltage-based Fingerprinting in Controller Area Networks," in *2023 IEEE 17th International Symposium on Applied Computational Intelligence and Informatics (SACI).* IEEE, 2023, pp. 000 231–000 236.

[27] B. Groza, A. Berdich, C. Jichici, and R. Mayrhofer, "Secure Accelerometer-Based Pairing of Mobile Devices in Multi-Modal Transport," *IEEE Access*, vol. 8, pp. 9246–9259, 2020.

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

[29] 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.

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

[31] A.-I. Radu and F. D. Garcia, "LeiA: A Lightweight Authentication Protocol for CAN," in *Computer Security–ESORICS 2016: 21st European Symposium on Research in Computer Security, Heraklion, Greece, September 26-30, 2016, Proceedings, Part II 21*. Springer, 2016, pp. 283–300.

[32] J. Schmandt, A. T. Sherman, and N. Banerjee, "Mini-MAC: Raising the bar for vehicular security with a lightweight message authentication protocol," *Vehicular Communications*, vol. 9, pp. 188–196, 2017.

[33] H. J. Jo, J. H. Kim, H.-Y. Choi, W. Choi, D. H. Lee, and I. Lee, "MAuth-CAN: Masquerade-Attack-Proof Authentication for In-Vehicle Networks," *IEEE transactions on vehicular technology*, vol. 69, no. 2, pp. 2204–2218, 2019.

[34] K.-D. Kang, Y. Baek, S. Lee, and S. H. Son, "An Attack-Resilient Source Authentication Protocol in Controller Area Network," in *2017 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*. IEEE, 2017, pp. 109–118.

[35] B. Palaniswamy, S. Camtepe, E. Foo, and J. Pieprzyk, "An Efficient Authentication Scheme for Intra-Vehicular Controller Area Network," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 3107–3122, 2020.

[36] S. Nürnberger and C. Rossow, "— vatiCAN — Vetted, Authenticated CAN Bus," in *Cryptographic Hardware and Embedded Systems–CHES 2016: 18th International Conference, Santa Barbara, CA, USA, August 17-19, 2016, Proceedings 18*. Springer, 2016, pp. 106–124.

[37] Q. Wang and S. Sawhney, "VeCure: A practical security framework to protect the CAN bus of vehicles," in *2014 International Conference on the Internet of Things (IOT)*. IEEE, 2014, pp. 13–18.

[38] H. Mun, K. Han, and D. H. Lee, "Ensuring safety and security in CAN-based automotive embedded systems: A combination of design optimization and secure communication," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 7, pp. 7078–7091, 2020.

[39] L. L. Bello, R. Mariani, S. Mubeen, and S. Saponara, "Recent Advances and Trends in On-Board Embedded and Networked Automotive Systems," *IEEE Trans. on Industrial Informatics*, vol. 15, no. 2, pp. 1038–1051, 2019.

[40] J. Liu, S. Zhang, W. Sun, and Y. Shi, "In-vehicle network attacks and countermeasures: Challenges and future directions," *IEEE Network*, vol. 31, no. 5, pp. 50–58, 2017.

[41] E. Aliwa, O. Rana, C. Perera, and P. Burnap, "Cyberattacks and Countermeasures for In-Vehicle Networks," *ACM Computing Surveys (CSUR)*, vol. 54, no. 1, pp. 1–37, 2021.

[42] A. Anwar, A. Anwar, L. Moukahal, and M. Zulkernine, "Security assessment of in-vehicle communication protocols," *Vehicular Communications*, p. 100639, 2023.

[43] W. Wu, R. Li, G. Xie, J. An, Y. Bai, J. Zhou, and K. Li, "A Survey of Intrusion Detection for In-Vehicle Networks," *IEEE Transactions on Intelligent Transportation Systems*, vol. 21, no. 3, pp. 919–933, 2019.

[44] O. Y. Al-Jarrah, C. Maple, M. Dianati, D. Oxtoby, and A. Mouzakitis, "Intrusion Detection Systems for Intra-Vehicle Networks: A Review," *IEEE Access*, vol. 7, pp. 21 266–21 289, 2019.

[45] C. Young, J. Zambreno, H. Olufowobi, and G. Bloom, "Survey of Automotive Controller Area Network Intrusion Detection Systems," *IEEE Design & Test*, vol. 36, no. 6, pp. 48–55, 2019.

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

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

[48] A. Tomlinson, J. Bryans, S. A. Shaikh, and H. K. Kalutarage, "Detection of Automotive CAN Cyber-Attacks by Identifying Packet Timing Anomalies in Time Windows," in *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*. IEEE, 2018, pp. 231–238.

[49] H. Olufowobi, C. Young, J. Zambreno, and G. Bloom, "SAIDuCANT: Specification-Based Automotive Intrusion Detection Using Controller Area Network (CAN) Timing," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 2, pp. 1484–1494, 2019.

[50] M. Al-Saud, A. M. Eltamaly, M. A. Mohamed, and A. Kavousi-Fard, "An Intelligent Data-Driven Model to Secure Intravehicle Communications Based on Machine Learning," *IEEE Transactions on Industrial Electronics*, vol. 67, no. 6, pp. 5112–5119, 2019.

[51] M. Marchetti and D. Stabili, "Anomaly detection of CAN bus messages through analysis of ID sequences," in *2017 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2017, pp. 1577–1583.

[52] S. Katragadda, P. J. Darby, A. Roche, and R. Gottumukkala, “Detecting Low-Rate Replay-Based Injection Attacks on In-Vehicle Networks,” *IEEE Access*, vol. 8, pp. 54 979–54 993, 2020.

[53] 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 *Industrial Networks and Intelligent Systems: 3rd International Conference, INISCOM 2017, Ho Chi Minh City, Vietnam, September 4, 2017, Proceedings 3*. Springer, 2018, pp. 285–294.

[54] D. Stabili, M. Marchetti, and M. Colajanni, “Detecting attacks to internal vehicle networks through Hamming distance,” in *2017 AEIT International Annual Conference*. IEEE, 2017, pp. 1–6.

[55] M. Markovitz and A. Wool, “Field classification, modeling and anomaly detection in unknown CAN bus networks,” *Vehicular Communications*, vol. 9, pp. 43–52, 2017.

[56] A. Taylor, S. Leblanc, and N. Japkowicz, “Anomaly Detection in Automobile Control Network Data with Long Short-Term Memory Networks,” in *2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*. IEEE, 2016, pp. 130–139.

[57] M. D. Hossain, H. Inoue, H. Ochiai, D. Fall, and Y. Kadobayashi, “LSTM-Based Intrusion Detection System for In-Vehicle Can Bus Communications,” *IEEE Access*, vol. 8, pp. 185 489–185 502, 2020.

[58] S. Longari, D. H. N. Valcarcel, M. Zago, M. Carminati, and S. Zanero, “CANnolo: An Anomaly Detection System Based on LSTM Autoencoders for Controller Area Network,” *IEEE Transactions on Network and Service Management*, vol. 18, no. 2, pp. 1913–1924, 2020.

[59] M.-J. Kang and J.-W. Kang, “Intrusion Detection System Using Deep Neural Network for In-Vehicle Network Security,” *PloS one*, vol. 11, no. 6, p. e0155781, 2016.

[60] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath *et al.*, “Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups,” *IEEE Signal processing magazine*, vol. 29, no. 6, pp. 82–97, 2012.

[61] C. E. Everett and D. McCoy, “OCTANE (Open Car Testbed and Network Experiments): Bringing Cyber-Physical Security Research to Researchers and Students,” in *Presented as part of the 6th Workshop on Cyber Security Experimentation and Test*, 2013.

[62] H. M. Song, J. Woo, and H. K. Kim, "In-vehicle network intrusion detection using deep convolutional neural network," *Vehicular Communications*, vol. 21, p. 100198, 2020.

[63] A. Alshammari, M. A. Zohdy, D. Debnath, and G. Corser, "Classification Approach for Intrusion Detection in Vehicle Systems," *Wireless Engineering and Technology*, vol. 9, no. 4, pp. 79–94, 2018.

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

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

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

[67] C. Dong, H. Wu, and Q. Li, "Multiple Observation HMM-based CAN bus Intrusion Detection System for In-Vehicle Network," *IEEE Access*, 2023.

[68] O. Y. Al-Jarrah, K. El Haloui, M. Dianati, and C. Maple, "A Novel Detection Approach of Unknown Cyber-Attacks for Intra-Vehicle Networks Using Recurrence Plots and Neural Networks," *IEEE Open Journal of Vehicular Technology*, vol. 4, pp. 271–280, 2023.

[69] K.-T. Cho and K. G. Shin, "Fingerprinting Electronic Control Units for Vehicle Intrusion Detection," in *25th USENIX Security Symposium (USENIX Security 16)*, 2016, pp. 911–927.

[70] X. Ying, S. U. Sagong, A. Clark, L. Bushnell, and R. Poovendran, "Shape of the Cloak: Formal Analysis of Clock Skew-Based Intrusion Detection System in Controller Area Networks," *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 9, pp. 2300–2314, 2019.

[71] S. U. Sagong, X. Ying, A. Clark, L. Bushnell, and R. Poovendran, "Cloaking the Clock: Emulating Clock Skew in Controller Area Networks," in *2018 ACM/IEEE 9th International Conference on Cyber-Physical Systems (ICCPS)*. IEEE, 2018, pp. 32–42.

[72] M. Tian, R. Jiang, C. Xing, H. Qu, Q. Lu, and X. Zhou, "Exploiting Temperature-Varied ECU Fingerprints for Source Identification in In-vehicle Network Intrusion Detection," in *2019 IEEE 38th International Performance Computing and Communications Conference (IPCCC)*. IEEE, 2019, pp. 1–8.

[73] P.-S. Murvay and B. Groza, “Source Identification Using Signal Characteristics in Controller Area Networks,” *IEEE Signal Processing Letters*, vol. 21, no. 4, pp. 395–399, 2014.

[74] P.-S. Murvay and B. Groza, “TIDAL-CAN: Differential Timing based Intrusion Detection And Localization for Controller Area Network,” *IEEE Access*, vol. 8, pp. 68 895–68 912, 2020.

[75] K.-T. Cho and K. G. Shin, “Viden: Attacker Identification on In-Vehicle Networks,” in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 1109–1123.

[76] 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 Transactions on Vehicular Technology*, vol. 67, no. 6, pp. 4757–4770, 2018.

[77] M. Kneib, O. Schell, and C. Huth, “EASI: Edge-Based Sender Identification on Resource-Constrained Platforms for Automotive Networks.” in *NDSS*, 2020, pp. 1–16.

[78] M. Wolf and R. Lambert, “Hacking Trucks – Cybersecurity Risks and Effective Cybersecurity Protection for Heavy Duty Vehicles,” *Automotive-Safety & Security 2017-Sicherheit und Zuverlässigkeit für automobile Informationstechnik*, 2017.

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

[80] R. Chatterjee, S. Mukherjee, and J. Daily, “Exploiting Transport Protocol Vulnerabilities in SAE J1939 Networks,” 2023.

[81] “J1939-71 – Vehicle Application Layer,” SAE International, Standard, May. 2006.

[82] N. Nowdehi, A. Lautenbach, and T. Olovsson, “In-Vehicle CAN Message Authentication: An Evaluation Based on Industrial Criteria,” in *2017 IEEE 86th Vehicular Technology Conference (VTC-Fall)*. IEEE, 2017, pp. 1–7.

[83] B. Palaniswamy, K. Ansari, A. G. Reddy, A. K. Das, and S. Shetty, “Robust Certificateless Authentication Protocol for the SAE J1939 Commercial Vehicles Bus,” *IEEE Transactions on Vehicular Technology*, 2022.

[84] J. Daily, D. Nnaji, and B. Ettlinger, “Securing CAN Traffic on J1939 Networks,” in *Workshop on Automotive and Autonomous Vehicle Security (AutoSec), 25 February 2021, Virtual*, 2021.

[85] "Secure TJA115x CAN Transceiver Family," https://www.nxp.com/products/interfaces/can-transceivers/secure-can-transceivers:SECURE-CAN, 2023, accessed: 2023-08-18.

[86] M. Zachos, "Securing J1939 Communications Using Strong Encryption with FIPS 140-2," in *SAE Technical Paper*. SAE International, 03 2017.

[87] J. Daily and P. Kulkarni, "Secure Heavy Vehicle Diagnostics," in *Proceedings of the Ground Vehicle Systems Engineering and Technology Symposium (GVSETS), NDIA*, 2020, pp. 13–15.

[88] "Introduction to RP1210A (RP1210B)," hhttps://www.kvaser.com/about-can/can-standards/rp1210/, 2023, accessed: 2023-08-19.

[89] S. Kumar, J. Daily, Q. Ahmed, and A. Arora, "Cybersecurity Vulnerabilities for Off-Board Commercial Vehicle Diagnostics," SAE Technical Paper, Tech. Rep., 2023.

[90] M. T. Campo, S. Mukherjee, and J. Daily, "Real-Time Network Defense of SAE J1939 Address Claim Attacks," *SAE International Journal of Commercial Vehicles*, vol. 14, no. 02-14-03-0026, pp. 319–328, 2021.

[91] "J1939-17 – CAN FD Physical Layer, 500 kbps/2 Mbps," SAE International, Standard, Dec. 2020.

[92] "J1939-91C – CAN FD Network Security," SAE International, Standard, November 2022.

[93] M. Mokhadder, M. Zachos, and J. Potter, "Evaluation of Vehicle System Performance of an SAE J1939-91C Network Security Implementation," SAE Technical Paper, Tech. Rep., 2023.

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

[95] N. Leslie, "An Unsupervised Learning Approach for In-Vehicle Network Intrusion Detection," in *2021 55th Annual Conference on Information Sciences and Systems (CISS)*. IEEE, 2021, pp. 1–4.

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

[97] M. Rogers, P. Weigand, J. Happa, and K. Rasmussen, "Detecting CAN Attacks on J1939 and NMEA 2000 Networks," *IEEE Transactions on Dependable and Secure Computing*, pp. 1–15, 2022.

[98] S. Hariharan, A. V. Papadopoulos, and T. Nolte, “On In-Vehicle Network Security Testing Methodologies in Construction Machinery,” in *IEEE 27th Intl. Conf. on Emerging Tech. and Factory Automation*, 2022, pp. 1–4.

[99] P. S. Oruganti, M. Appel, and Q. Ahmed, “Hardware-in-loop based Automotive Embedded Systems Cybersecurity Evaluation Testbed,” in *Proceedings of the ACM Workshop on Automotive Cybersecurity*, 2019, pp. 41–44.

[100] H. Zhang, K. Huang, J. Wang, and Z. Liu, “CAN-FT: A Fuzz Testing Method for Automotive Controller Area Network Bus,” in *2021 International Conference on Computer Information Science and Artificial Intelligence (CISAI)*. IEEE, 2021, pp. 225–231.

[101] M. Dunne and S. Fischmeister, “Powertrace-based Fuzzing of CAN Connected Hardware,” in *2022 IEEE International Conference on Cyber Security and Resilience (CSR)*. IEEE, 2022, pp. 239–244.

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

[103] “ISO11898-3. Road vehicles — Controller area network (CAN) — Part 3: Low-speed, fault-tolerant, medium-dependent interface,” International Organization for Standardization, Standard, 1st edition, Jun 2006.

[104] “ISO11898-4. Road vehicles — Controller Area Network (CAN) — Part 4: Time-triggered communication,” International Organization for Standardization, Standard, 1st edition, Aug 2004.

[105] “Industrial Automation and Control using CAN Protocols,” https://www.elprocus.com/industrial-automation-control-using-can-protocol/, 2023, accessed: 2023-08-13.

[106] “SIMPLIFY 3D PRINTER WIRING WITH CAN BUS,” https://hackaday.com/2021/11/03/simplify-3d-printer-wiring-with-can-bus/, 2023, accessed: 2023-08-13.

[107] “CAN Protocols,” https://sterlingmedicaldevices.com/our-work/medical-device-projects/can-protocols/, 2023, accessed: 2023-08-13.

[108] P.-S. Murvay and B. Groza, “A brief look at the security of DeviceNet communication in industrial control systems,” in *Proceedings of the Central European Cybersecurity Conference 2018*, 2018, pp. 1–6.

[109] N. Su, X. Tu, Q. Yang, X. Li, and Y. Tong, “Design of Data Communication and Monitoring System Based on Aviation ARINC825 Bus,” in *2019 IEEE 11th International Conference on Communication Software and Networks (ICCSN)*, 2019, pp. 286–290.

[110] “Investigating CAN Bus Network Integrity in Avionics Systems,” https://www.rapid7.com/research/report/investigating-can-bus-network-integrity-in-avionics-systems/, 2023, accessed: 2023-08-13.

[111] “J1939-13 – Off-Board Diagnostic Connector,” SAE International, Standard, March 2004.

[112] “J1939-21 – Data Link Layer,” SAE International, Standard,, Dec. 2006.

[113] “Digital Annex of Serial Control and Communication,” SAE International, Standard, January. 2020.

[114] “J1939-81 – Network Management,” SAE International, Standard, March. 2003.

[115] C. Jichici, “An IDS for CAN bus using neural networks,” Master’s thesis, University Politehnica of Timisoara, 2018.

[116] H. Lee, S. H. Jeong, and H. K. Kim, “OTIDS: A Novel Intrusion Detection System for In-vehicle Network by Using Remote Frame,” *Privacy, Security and Trust (PST) 2017*, 2017.

[117] *Programming With CAPL*, Vector, December. 2004.

[118] D. Dolev and A. Yao, “On the security of public key protocols,” *IEEE Transactions on information theory*, vol. 29, no. 2, pp. 198–208, 1983.

[119] B. Groza, L. Popa, P.-S. Murvay, Y. Elovici, and A. Shabtai, “CANARY – a reactive defense mechanism for Controller Area Networks based on Active RelaYs,” in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 4259–4276.

[120] B. Groza, L. Popa, T. Andreica, P.-S. Murvay, A. Shabtai, and Y. Elovici, “PanoptiCANs-Adversary-Resilient Architectures for Controller Area Networks,” in *European Symposium on Research in Computer Security*. Springer, 2022, pp. 658–679.

[121] K.-T. Cho and K. G. Shin, “Error Handling of In-vehicle Networks Makes Them Vulnerable,” in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2016, pp. 1044–1055.

[122] 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, 2017, pp. 71:1–71:10.

[123] *CAPL DLL Description*, Vector, 2007.

[124] M.-Y. Su, “Real-time anomaly detection systems for Denial-of-Service attacks by weighted k-nearest-neighbor classifiers,” *Expert Systems with Applications*, vol. 38, no. 4, pp. 3492–3498, 2011.

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

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

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

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

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

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

[131] S. Jain and J. Guajardo, “Physical Layer Group Key Agreement for Automotive Controller Area Networks,” in *International Cryptographic Hardware and Embedded Systems*. Springer, 2016, pp. 85–105.

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

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

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

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

[136] “FMS-Standard description,” ACEA Task Force HDEI/BCEI, Standard, September 2021.

[137] A. A. Cárdenas, S. Amin, Z.-S. Lin, Y.-L. Huang, C.-Y. Huang, and S. Sastry, “Attacks against process control systems: risk assessment, detection, and response,” in *ACM symposium on information, computer and communications security*, 2011, pp. 355–366.

[138] E. S. Page, “Continuous Inspection Schemes,” *Biometrika*, vol. 41, no. 1/2, pp. 100–115, 1954.

[139] M. Ziv, “CANCAN: Encapsulation of CAN-FD Messages for Circumvention of Security Measures,” CYMOTIVE Technologies LTD, Report, June. 2022.

[140] A. Kavousi-Fard, T. Jin, W. Su, and N. Parsa, “An Effective Anomaly Detection Model for Securing Communications in Electric Vehicles,” *IEEE Transactions on Industry Applications*, pp. 1–1, 2020.

[141] M. Al-Saud, A. M. Eltamaly, M. A. Mohamed, and A. Kavousi-Fard, “An Intelligent Data-Driven Model to Secure Intravehicle Communications Based on Machine Learning,” *IEEE Transactions on Industrial Electronics*, vol. 67, no. 6, pp. 5112–5119, 2019.