PhD-FSTM-2022-018

Faculty of Science, Technology and Medicine

DISSERTATION

Presented on 04/03/2022 in Esch-sur-Alzette, Luxembourg

to obtain the degree of

Docteur de l'Université du Luxembourg

en Sciences de l'Ingeniéur

by

Alessio Buscemi

Born on 02/08/1994 in Rome, Italy

Automation of Controller Area Network Reverse Engineering: Approaches, Opportunities and Security Threats

UNIVERSITÉ DU LUXEMBOURG

Dissertation Defence Committee:

Prof. Dr. Ulrich Sorger *Chair*	Department of Computer Science, University of Luxembourg
Prof. Dr. Andriy Panchenko *Vice-Chair*	Institute of Computer Science, Brandenburg University of Technology
Prof. Dr. Thomas Engel *Supervisor*	Department of Computer Science, University of Luxembourg
Prof. Dr. Kang Shin	Electrical Engineering and Computer Science Department, University of Michigan
Prof. Dr. Falko Dressler	School of Electrical Engineering and Computer Science, TU Berlin

Advisors:

Dr. Ion Turcanu	IT for Innovative Services Department, Luxembourg Institute of Science and Technology
Dr. German Castignani	Motion-S S.A.

Affidavit

I hereby confirm that the PhD thesis entitled "Automation of Controller Area Network Reverse Engineering:Approaches, Opportunities and Security Threats" has been written independently and without any other sources than cited.

Luxembourg, ____________________ ____________________________

Name

Acknowledgements

This doctorate would not have been possible without Prof. Engel who, during my Erasmus exchange at the University of Luxembourg, introduced me to the research opportunities offered by the SECAN-Lab. His guidance and willingness to discuss research initiatives was an essential resource in completing this PhD. My sincere thanks go to him.

I would like to extend my deepest gratitude to my supervisor and mentor, Dr. Turcanu, for his continuous teaching and support. His suggestions, reviews and criticisms have been a vital resource in my academic path. He gave me direction and methodology, and let me grow through my mistakes, which ultimately allowed me to conduct independent research. Other honorable mentions go to Dr. Castignani and Prof. Panchenko, whose guidance, feedback and insights have greatly helped me enrich my research output.

I wish to acknowledge the invaluable assistance of Xee and, in particular, of Dr. Crunelle and Mr. Castronovo, not only for providing me with the means to carry out most of my research, but also for their insights into the automotive industry and CAN reverse engineering. I would also like to thank Mr. Stemper for his precious help in the data collection process.

In a personal capacity, I am infinitely grateful to my girlfriend Aisha and my parents for their constant and unconditional support, which was essential to face the - many - difficult times of the last two years. I often wonder if this achievement would have been possible without them.

Special thanks go to Massimo, Laura, Giovanni, Giorgio, Antonio, Mario Giulio and Orso for making their cars available for data collection despite the initial fear that I would mess with their electronic system. Finally, I would like to thank the colleagues and staff of SECAN-Lab, the ESN Luxembourg team and all my friends here in Luxembourg for enriching this chapter of my life with memorable lessons and experiences.

Index

List of Figures

List of Tables

List of Algorithms

Abstract

Controller Area Network (CAN) is the de-facto in-vehicle communication system in the automotive industry today. CAN data represents a valuable source of information regarding the vehicle, which can be exploited for a multitude of purposes by aftermarket companies, from fleet management to infotainment. With the rise of Vehicular Ad Hoc Networks (VANETs) and autonomous driving, we can expect the amount of data transiting on the CAN bus to further augment in the near future. While not encrypted, the communication inside the CAN bus is typically encoded using proprietary formats of the Original Equipment Manufacturers (OEMs) in order to prevent easy access to the information exchanged on the network. However, given the unwillingness of the OEMs to disclose the formats of most of the CAN signals of commercial vehicles (cars in particular) to the general public, the most common way to obtain such information is through *reverse engineering*.

Recently, researchers have started investigating the automation of this process to make it faster, scalable and standardised. Aside from the evident advantages that it would bring to the industry, the automation of CAN bus reverse engineering has also gained interest in the scientific community, where automotive cybersecurity is a prominent topic. While achieving convincing results, the automation of CAN reverse engineering is still invasive, often includes complex hardware configurations or requires the presence of a human operator in the vehicle. This dissertation aims to analyse the main advancements achieved in the field of CAN bus reverse engineering and shed light on open issues.

In the first part of this dissertation, we explore opportunities and challenges of the automation of CAN bus reverse engineering and present three approaches that achieve different degrees of automation. The first, FastCAN, is based on the taxonomy of signals. Its goal is to provide a complete, standardised and modular pipeline for semi-automated reverse engineering and reduce the total time for data collection. The second, CSI, is a Machine Learning (ML)-based algorithm for the identification of critical signals working under limited assumptions. We use CSI as a case study to investigate whether CAN reverse engineering can be achieved

with no other hardware than a dongle for the collection of raw data. The third, CANMatch, is a complete and fully automated approach based on frame matching. Through CANMatch we seek to demonstrate that the reuse of CAN frame IDs can be exploited to reverse engineer a high number of signals with minimal hardware requirements and human effort.

In the second part of this dissertation, we discuss the implications that the full automation of the reverse engineering process has on the security of the bus. In this context, we investigate whether the anonymisation of the CAN frame IDs is sufficient to prevent frame-matching based reverse engineering. The results highlight that ML models can fingerprint CAN frames despite the anonymisation of their IDs. Finally, we propose a defence against frame fingerprinting based on traffic mutations, such as padding on the payload and morphing on the sending frequency. We conclude that traffic mutations are a promising study direction to prevent frame-matching based reverse engineering.

Bibliographic notes

This dissertation is based on the following jointly authored publications:

- **Section 5**: "Poster: A Methodology for Semi-Automated CAN Bus Reverse Engineering" with Ion Turcanu, German Castignani, Romain Crunelle and Thomas Engel, which appeared at the IEEE Vehicular Networking Conference (VNC), 2021 [1].
- **Section 6**: "A Data-Driven Minimal Approach for CAN Bus Reverse Engineering" with German Castignani, Thomas Engel and Ion Turcanu, which appeared at the IEEE 3rd Connected and Automated Vehicles Symposium (CAVS), 2020 [2].
- **Sections 4 and 7**: "CANMatch: A Fully Automated Tool for CAN Bus Reverse Engineering based on Frame Matching" with Ion Turcanu, German Castignani, Romain Crunelle and Thomas Engel, published in IEEE Transactions on Vehicular Technology, vol. 70, no. 12, 2021 [3].
- **Section 10**: "On Frame Fingerprinting and Controller Area Networks Security in Connected Vehicles" with Ion Turcanu, German Castignani and Thomas Engel, which appeared at the IEEE Consumer Communications & Networking Conference, 2022 [4].

Part I

INTRODUCTION

1 Dissertation Overview

In the last decades, the innovation in the automotive sector has been mainly driven by the electrification and digitalisation of the internal components [5], [6]. While in 1950s the electronic components represented only around 1 % of the production costs of commercial vehicles, this percentage has grown to 35 % in 2018, and it is expected to rise to 50 % by 2030 [7]. Furthermore, the global market of automotive data, of which in-vehicle data is a major share, is estimated to be worth between $450-750 billion in 2030 [8]. In-vehicle data is an essential asset for researchers and companies who work on a variety of automotive solutions, such as driver profiling, fleet management, cloud services, pooling etc. [9]–[13].

The electronic sensors embedded in a vehicle, known as Electronic Control Unit (ECU) [14], typically communicate to each other through a physical network. Due to the stringent requirements of in-vehicle communications, there is only a limited number of network systems that can be currently found in commercial vehicles around the world, such as Controller Area Network (CAN), Local Interconnect Network (LIN), FlexRay, Media Oriented Systems Transport (MOST), and Automotive Ethernet [15]. Among these, CAN is considered the de-facto world standard for the internal communication of commercial vehicles and trucks. When multiple network systems are present in a vehicle, they are typically designed as sub-networks connected to a central CAN in the powertrain.

Released in 1991 by Bosch GmbH [16], the version 2.0 of CAN bus [17]–[19] has quickly risen to fame thanks to its unique features, namely the low production costs and the high robustness to electrical noise. Aside from its wide usage in the automotive sector, nowadays CAN is integrated in a variety of industries, such as production of elevators and escalators, medical equipment, industrial control, ships, railroads, equipment for aviation and navigation [20].

CAN is a peer-to-peer serial bus, which allows ECUs to send and receive data without the need of an orchestrator node. The data exchange is based on messages, or *frames*, typically carrying one or multiple signals. Signals encapsulate values of telemetries and vehicle functions, which describe the real-time status of a vehicle. The CAN protocol does not include any security features, such as encryption of data payload or ECU authentication. Nonetheless, interpreting the data by simply logging the traffic is not straightforward. As a matter of fact, signals are encoded according to a proprietary format by each Original Equipment Manufacturer (OEM), which is not accessible to the general public.

A few solutions are currently available to address the need of interpretable CAN data. Namely, a set of standards under the name SAE J1939 [21] aim to provide a standardised data stream between ECUs. SAE J1939 has gained popularity in the sector of heavy duty diesel vehicles, to the point that nowadays most of them are equipped with it [22]. Nonetheless, OEMs are still reluctant about granting complete access to the commercial cars. For this reason, as of today, the only available option to interpret CAN signals in vehicles remains reverse engineering.

CAN bus reverse engineering is the process of identifying signal boundaries within frames and decode their semantic meaning and format. In recent years, the optimisation and automation of this process has caught the attention of the scientific community and automotive companies. A fast, standardised and reliable reverse engineering is considered an essential requirement to conduct research and develop products based on CAN data.

Many solutions for the automation of CAN bus reverse engineering have been presented in the literature. Some solutions provide a complete structured pipeline, from data collection to translation of the signals' format [23]–[26], while others focus on specific aspects, e.g., tokenisation [27]–[29]. These works exploit different combinations of hardware equipment, but also algorithms from a variety of domains, from combinatorial optimization [24], [30] to

Machine Learning (ML) [31]–[33]. As a consequence, the field of CAN reverse engineering appears heterogeneous and difficult to approach by non-experts.

A number of wired and wireless attacks based on the injection of malevolent messages in the CAN bus have raised concern in recent years [34]–[36]. In 2015, Miller and Valasek [36] caused public stir when they took remote control of a Jeep Cherokee (and drove it off the road) by inputting CAN messages through the Telematic Control Unit (TCU) and In-Vehicle Infotainment (IVI) system. In this context, the automation of the CAN bus reverse engineering process plays an essential role. On the one hand, it raises questions about the ease with which adversaries can conduct attacks [37]. On the other hand, it offers a fast and standardised solution that allows researchers who work on intrusion detection to get access to interpretable in-vehicle data [38]–[41].

The main motivation for this dissertation is to aid readers from the scientific community and the automotive industry in developing a methodical understanding of the process, opportunities, challenges and security issues of automated CAN bus reverse engineering.

1.1 Research Questions

In the recent years, the interest of the scientific community and the industry towards in-vehicle data and, more specifically, CAN data, has grown steadily. Despite the efforts made by researchers in easing the interpretation of such data, CAN reverse engineering is still perceived as a complex task.

The first part of this work is an investigation of the challenges posed by the automation of CAN reverse engineering. In particular, we aim at identifying the core requirements and steps that characterise this process. We conduct a study of the opportunities offered by the state-of-the-art methodologies proposed in literature and their limitations with the purpose of determining areas of improvement. This research on CAN bus reverse engineering is driven by two main questions:

1. *What is the best trade-off between the achieved performance and the needed requirements?*

2. *How much does the diversity in the type and number of ECUs attached to the CAN bus*

affect the performance of the reverse engineering?

The second part of this dissertation is dedicated to the understanding of the security aspects revolving around the reverse engineering of the CAN bus. The advent of Vehicular Ad Hoc Networks (VANETs) [42]–[44] and Connected and Automated Vehicles (CAVs) [45]–[47] is driving an increase of wireless access points to in-vehicle networks and, thus, the growth of the surface exploitable remotely by potential attackers. In this context, the automation and easing of the reverse engineering process facilitates the access of the attackers to clear in-vehicle data, thus posing non-negligible risks to the security of the vehicles and the safety of the passenger. In this dissertation, three fundamental questions related to these aspects are addressed:

1. *Can the reverse engineering be performed entirely from remote without prior knowledge of the vehicle?*
2. *Does the automation of reverse engineering incentivise new attacks?*
3. *What are suitable countermeasures against CAN reverse engineering?*

1.2 Methodology

The first concern of researchers working on CAN is the acquisition of in-vehicle data. CAN data can be usually logged through the On-Board Diagnostics (OBD-II) interface with a commercial dongle [48]–[50] or directly to the physical wires. The purpose of the research on CAN reverse engineering is to develop a universal solution applicable on any vehicle model. For this reason, the dataset of logs employed for the study and the evaluation of CAN decoding methodologies should be large and highly diverse in terms of model, manufacturer, market segment and year of production.

The evaluation of four out of five studies presented in this dissertation is partially or fully based on a diverse dataset of logs from 477 distinct vehicle models belonging to 28 different OEM, acquired from an industrial partner, Xee [51], a car telematics company expert in manual reverse engineering. These logs are 10 s long and have been collected during a 5 years span, 2016-2020, throughout sessions in which the vehicles are inactive, i.e. they are parked and no operation is carried out by a human operator. Due to their short length and the context in which they were logged, these traces are not suitable to conduct certain experiments, e.g. on tokenisation. As a consequence, additional logs related to 15 distinct vehicle models were collected by us between May 2020 and July 2021. These logs are diverse in terms of drivers and driving scenarios.

The second concern when conducting any study on CAN and, in particular, on reverse engineering, is the ground truth employed for the validation. Since the proprietary format to interpret CAN data is kept secret by the OEMs, the only available approach is to validate the proposed methodology against a ground truth generated through manual reverse engineering (see Section 2.5). In the studies conducted by us and presented in this dissertation, the results of the manual reverse engineering obtained by expert technicians at Xee were employed. For further validation, two logs were also retrieved from OpenDBC, an open repository by CommaAI [52]. The ground truth obtained in such fashion covers all the vehicle models tested in the studies.

CAN bus reverse engineering is a topic heavily tied with data science. For this reason, aspects and methodologies from this macro subject are considered in this work. Namely, a number of data science techniques are adopted for the pre-proprocessing of CAN raw input data and feature engineering, such as time series interpolation and *smoothing*, *sliding windows*, outliers removals, Synthetic Minority Over-sampling TEchnique (SMOTE) etc. Supervised ML algorithms, such as Random Forest (RF), Neural Network (NN), Support Vector Machine (SVM) etc., are employed for the fingerprinting and the identification of CAN signals and frames. In this scope, both Closed Set and Open Set Recognition (OSR) assumptions are considered. Furthermore, regressions are used to decode the format of the signals, while a unsupervised ML algorithm, Density-Based Spatial Clustering of Applications with Noise (DBSCAN), is applied to cluster semantically similar signals.

All these algorithms and, in particular, those related to the engineering of the features, are adapted to take into consideration the unique properties and constraints of the CAN bus, such as data syntax and semantics, bandwidth, frames sending frequency etc.

1.3 Contributions

The first contribution of this work is a structured comparative study of state-of-the-art methodologies for CAN bus reverse engineering (Section 3). These methodologies are grouped based on the underlying approach that characterises them. We highlight advantages and disadvantages in adopting each of these approaches.

The second contribution is an improved version of the Reverse Engineering of Automotive Data frames (READ) [28] algorithm for tokenisation, which increases the tokenisation performance by identifying the signal endianness (Section 4).

The third contribution is FastCAN, a tool for the optimisation of semi-automated CAN reverse engineering (Section 5). Related work has proven that semi-automated reverse engineering preserves the reliability of the manual approach while reducing the time needed for the data collection and decoding [23], [53]. FastCAN is a non-invasive methodology for semi-automated CAN bus reverse engineering, whose main novelties are (i) the division of the data collection in multiple short sessions targeting specific groups of signals and (ii) the translation driven by the taxonomy of the signals.

The fourth contribution is Critical Signals Identifier (CSI), a ML-based algorithm for the decoding of the semantic meaning of signals considered critical for the functioning of the vehicle (Section 6). Extending the work of [54], CSI extracts intrinsic features of the critical signals based on a few seconds traces to identify them in a larger pool of noise signals. The main novelties introduced by CSI are the division of signals into categories based on their length, the use of categorical features to represent CAN signals, and the employment of Integer Linear Programming (ILP) to refine the predictions made by the ML models.

The fifth contribution is CANMatch, a framework for automated CAN bus reverse engineering that is able to decode the format of an unknown vehicle by exploiting frame ID reuse (Section 7). Differently from other proposed solutions – which require the injection of diagnostic messages, the usage of external Inertial Measurement Unit (IMU)/Global Positioning System (GPS) data, expertise and partial manual effort – CANMatch gets in input only the CAN raw trace of the target vehicle to reverse engineer and clear information on the formats adopted in other vehicle models. The novelties introduced by CANMatch are (i) the reuse of frames across different vehicle models to achieve a fully automated CAN bus reverse engineering; (ii) a clustering-based method that identifies the redundant signals – i.e., carrying the same vehicle functions (although, possibly encoded in a different format) – by finding correlations between previously decoded signals and undecoded tokens.

The sixth contribution is a study about whether the abandonment of the frame ID reuse practice is sufficient to completely anonymise the CAN frames, thus nullifying the benefit of a reverse engineering approach based on frame matching (Section 10). In particular, the possibility of performing matching on anonymised frames by exploiting other properties of the frames is investigated. We present a new ML-based algorithm following OSR assumptions, whose goal is to deanynonymise frames.

The seventh contribution is a countermeasure against the matching of frames based on traffic mutations, i.e. altering the traffic in such a way that the resultant distribution of values over certain properties of the frames becomes more similar among different frames (Section 11). The goal of such operations on the CAN traffic is to reduce the recognisability of frames, thus making the ID anonymisation an effective defence against frame matching-based fingerprinting. Differently from other proposed solutions for CAN security, this methodology assumes that no changes to the CAN protocol are applied and no external hardware is attached to the bus and the ECUs.

The last contribution is an extensive evaluation of possible directions for future research (Section 12). This includes proposals for improving the performance of the state-of-the-art reverse engineering methodologies, suggestions for decoding multiplexed CAN data, and a discussion about suitable defences against reverse engineering.

1.4 Manuscript Structure

In Part II, we provide a background on in-vehicle networks and CAN bus reverse engineering. In Section 2, we introduce and benchmark the main networks adopted for in-vehicle communication. In this context, we describe the characteristics of the CAN protocol and the CAN data. In Section 3, we introduce reverse engineering and contextualise our work accordingly. In this scope, we review extensively the state-of-the-art CAN reverse engineering approaches.

Part III contains the details of our work on CAN reverse engineering. In Section 5, we present FastCAN, our novel tool to perform semi-automated CAN bus reverse engineering. Section 6 describes CSI, a methodology for fully automated reverse engineering of a number of signals considered critical for the functioning of the vehicle. In Section 7, we propose CANMatch, a tool for fully automated reverse engineering based on frame matching. In Section 8, we benchmark FastCAN, CSI and CANMatch against the state-of-the-art CAN reverse engineering solutions presented in the literature.

In Part IV, we present our research on preventing frame matching-based fully automated reverse engineering. It starts with Section 9, in which the problem of the lack of encrypted communication in CAN is introduced, as well as the implications for the security and safety of the vehicles. In Section 10, we investigate whether the anonymisation of the frame IDs is sufficient to prevent the reverse engineering. In Section 11, we present a countermeasure against frame fingerprinting based on mutations of the CAN traffic.

Part V is the conclusive part of the dissertation. In Section 12, we explore possible directions for future research in the field of CAN bus reverse engineering and security. In Section 13, all contributions contained in this dissertation are summarised and final thoughts regarding the presented work are provided.

Part II

BACKGROUND

2

Introduction to CAN

In this section, we introduce the networks employed for the internal communication of vehicles, with a focus on CAN. Subsequently, we describe more in detail CAN data and CAN data collection.

2.1 In-Vehicle Networks

The backbone of all electrical systems in modern vehicles is intra-vehicle networking. The sustained growth in the number of the ECUs, as well as new design requirements led by the evolving needs of the automotive market, has brought to the release of a number of in-vehicle networks, the most notable being CAN, Controller Area Network Flexible Data-Rate (CAN FD), LIN, FlexRay, MOST and Automotive Ethernet. Such networks are diverse in terms of characteristics and also employed for different purposes. In the following, we describe and compare these networks. The purpose is to highlight the strengths and the weaknesses of the CAN bus with respect to other networks widely embedded in commercial vehicles.

2.1.1 CAN

CAN is a master-less message-based serial bus standard for the communication of ECUs. The CAN protocol covers the first two layers of the Open Systems Interconnection (OSI) model

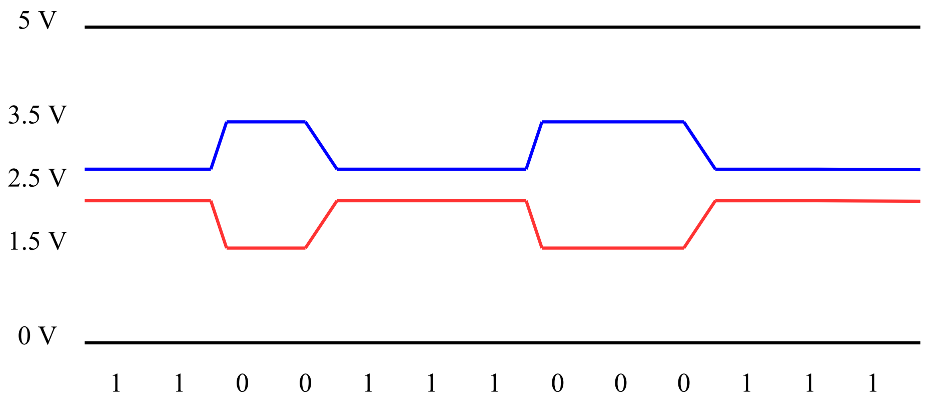


Figure 1: An example of data transmission on a High Speed CAN.

[55], i.e., physical and data-link.

At the physical layer, it consists of two twisted wires, CAN High (CANH) and CAN Low (CANL), with a nominal characteristic impedance of 120 Ω. The data is transmitted as differential wired-AND signals. There are two implementations of the physical layer of CAN 2.0: the *High Speed* (ISO 11898-2 [18]) and *Low Speed* (ISO 11898-3 [19]). The High Speed CAN baud rate ranges from 40 kbit/s to 1 Mbit/s (depending on the length of the bus). The Low Speed CAN is characterised by baud rates ranging from 40 kbit/s to 125 kbit/s.

In the High Speed CAN, to transmit a dominant state, the wires reach a differential voltage of 2 V, with the CANH and CANL being driven, respectively, towards 3.5 V and 1.5 V. To transmit the recessive state, the wires must have a differential voltage of less than 0.5 V. Similarly to High Speed CAN, in Low Speed CAN the wires must reach a differential voltage of 2 V around 2.5 V to signal a dominant state. On the contrary, the recessive state is transmitted when the CANH and CANL are pushed, respectively, towards 5 V and 0 V. This standard is also known as Fault Tolerant, since it achieves fault tolerance with respect to wiring failures. At bit-level, 0 represents the dominant state, while 1 encodes the recessive state. Figure 1 illustrates an example of data transmission on a High Speed CAN.

At the Data-Link layer (ISO11898-1 [17]), the communication on the CAN bus relies on messages (or frames). The frames do not carry the address of the sending nor of the receiving ECU. Due to the absence of a master node in charge of orchestrating the communication, if

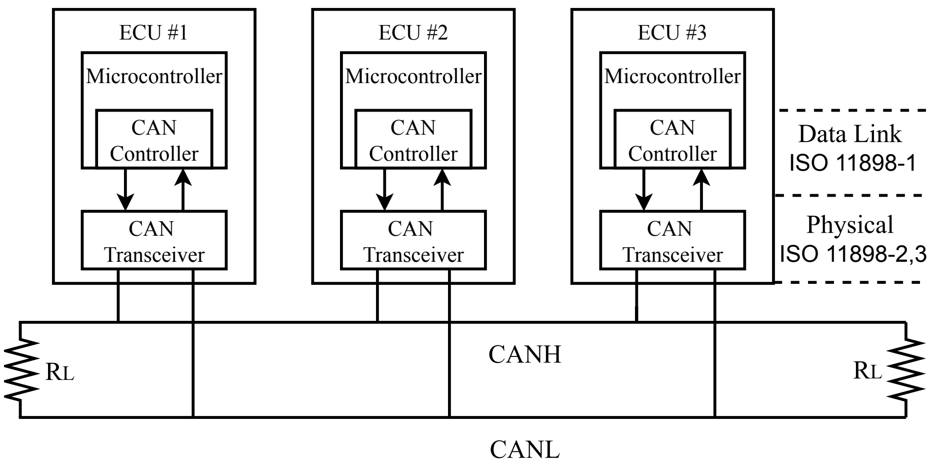


Figure 2: An example of CAN bus with three ECUs attached.

the bus is free, a frame is received by all the other ECUs. On the contrary, when multiple messages are sent concurrently, a collision occurs. The resolution of collisions in CAN bus is handled through an arbitration process. The frame with the highest priority ID overcomes the ones with lower priorities. This is achieved through the use of dominant bits overwriting the recessive ones. It follows that high priority IDs are those with a low value, while low priority IDs have high values. The ECU sending the dominant frame completes the transmission, while the rest of the ECUs turn into receivers. When the bus is free again, the ECUs which lost the arbitration will attempt to retransmit their messages. Figure 2 shows the essential elements that compose a CAN bus.

A CAN frame is composed of several fields:

- **Start of frame** – 1 bit
- **Identifier (ID)** – 11 bit or 29 bit in the extended version – International Organization for Standardization (ISO) 11898-1 [56]. It identifies the frame and denotes its priority.
- **Remote Transmission Request (RTR)** – 1 bit, it is dominant or recessive for, respectively, data and remote request frames.
- **Identifier extension bit (IDE)** – 1 bit, it is dominant or recessive for, respectively, for the standard 11-bit identifier and the extended version.

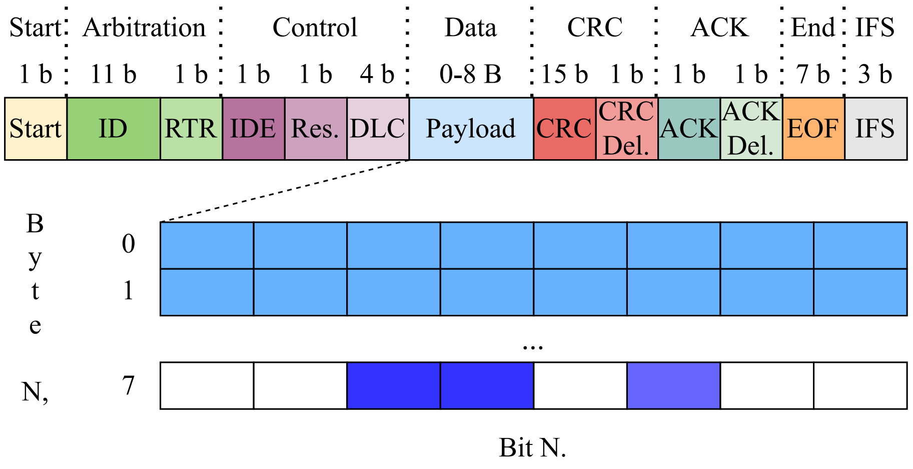


Figure 3: Example of CAN frame with 8-bytes payload. Each color in the payload represents a different signal.

- **Reserved** – 1 bit, reserved for future use.
- **Data Length Code (DLC)** – 4 bit, it indicates the length of the payload, expressed in bytes.
- **Payload** – between 0–8 Byte, it carries the actual content of the frame, encapsulated in bits.
- **Cyclic Redundancy Check (CRC)** – 15 bit
- **Acknowledge (ACK)** – 1 bit, recessive when the frame is sent.
- **End of Frame (EOF)** – 7 bit
- **Inter-frame spacing (IFS)** – 3 bit, must be recessive.

Figure 3 shows an example of a CAN frame with a payload of 8 Byte.

2.1.2 CAN FD

Released in 2012 by Bosch GmbH, CAN Flexible Data-Rate (FD) – ISO11898-1:2015 [17] – is an enhanced version of the CAN protocol. It was designed to meet the needs of the automotive industry for a higher bandwidth to support the steady increase in the number of ECUs present in vehicles. The main advantage of CAN FD compared to the original CAN is the dual bit-rate. While the bit rate of the arbitration phase is the same, the payload can be transmitted at a higher bit rate, allowing the payload length to increase from a maximum of 8 Byte to 64 Byte. As a consequence, CAN FD achieves a superior communication bandwidth up to 5 Mbit/s (with a 40 m long bus). In addition, an improvement in the CRC field and algorithm makes CAN FD more reliable. Related work on reverse engineering has solely focused on CAN 2.0. Hence, hereafter CAN will refer only to the version 2.0, unless specified otherwise.

2.1.3 LIN

LIN – ISO 17987-1 [57]– is a serial network protocol released in 2002. It was designed by a consortium of five automakers to address the need of a standardised cheap serial bus for non-critical body functions, e.g. windows, wipers etc. It is typically employed for small sub-systems that are connected to the main CAN network, thus creating a hierarchy of vehicle networks.

It is based on a single wire and provides a communication bandwidth up to 19.2 kbit/s with a bus length of 40 m. It is a broadcast network composed of up to 16 nodes, i.e. one master and up to 15 slaves. The master initiates the messages, which are replied by at most one slave, including the master itself that can act as one. Given that the master orchestrates the communication, no collision detection mechanism is implemented. The communication relies on messages with a length of 2, 4 or 8 Byte. The access to the channel is regulated through Carrier-Sense Multiple Access with Collision Avoidance (CSMA/CA) [58].

2.1.4 FlexRay

FlexRay – ISO 17458 [59] – is a fast and highly reliable bus released by the FlexRay Consortium in 2006, which addresses the growing needs for safety-critical applications in the automotive sector. It is characterised by a communication bandwidth up to 10 Mbit/s and can be implemented as party line or star bus topologies. It can support two independent data channels for fault tolerance, which allows the bus the operate with limited bandwidth if one of

Table 1: Comparison of different versions of MOST

	MOST25	**MOST50**	**MOST150**
Bit rate	$\sim$ 25 Mbps	$\sim$ 50 Mbps	$\sim$ 150 Mbps
Physical Layer	Optical	Electrical, Optical	Electrical, Optical

the channels becomes inoperative. Its communication is based on a time cycle, which consists of a *static segment* and a *dynamic segment*.

The static segment ensures bounded latency, as the nodes can transmit messages according to fixed length time slots preliminarily assigned to them, according to the Time Division Multiple Access (TDMA) method [60]. The communication schedule is handled through synchronised local counters.

The dynamic segment is based on the Frequency-Time Division Multiple Access (FTDMA) algorithm [61] and is used to address changing bandwidth requirements at run time. In this case, the messages can be transmitted by the nodes when needed. Given that the dynamic segment has a finite length, the protocol does not guarantee that the messages of the nodes are sent during the time cycle.

2.1.5 MOST

MOST – ISO21806 [62] – is a master-slave bus designed in 1997, used for the transmission of high quality of service multimedia content, such as voice, audio and video. It is a Time Division Multiplexing (TDM) network, typically based on a ring topology. Other configurations, such as virtual star or double ring (for safety critical applications), are allowed as well. It connects up to 64 devices, that can be effortlessly attached and detached. MOST comes in three versions characterised by different bit rates, MOST25, MOST50 and MOST150, presented in Table 1 [63].

The specifications of MOST include all 7 layers of the ISO/OSI model. The communication can be synchronous, asynchronous or isochronous and it is based on three types of data (i) *common packet*, (ii) *streaming* and (iii) *control*. The different types of data transit over up to 60 different channels and do not interfere with each other. The role of the master node, typically referred to as *timing node*, is to continuously send frames on the network. The slaves, also known as *timing followers*, synchronise through a preamble transmitted at the beginning of each message transfer. The channel access is handled through TDMA or CSMA/CA.

2.1.6 Automotive Ethernet

Ethernet is a protocol standardised in 1980 by Institute of Electrical and Electronics Engineers (IEEE) [64] and the de-facto global standard for the wired connection of computers and many electronic devices. Given its large success for a variety of applications, the automotive sector has investigated the adoption of Ethernet as in-vehicle network in the last years – ISO21111 [65]. Its versatility, the low cost for the production, the low weight of the wiring harnesses – a single unshielded copper twisted pair is used in the automotive implementation – make Ethernet a good candidate for in-vehicle networking. There are two main standards for the physical layer – ISO21111 [66] –, 100BASE-T1 (IEEE 802.3bw), with a nominal speed of 100 Mbps, and IEEE 1000BASE-T1 (IEEE 802.3bp), with a nominal speed of 1000 Mbps. In Ethernet, the communication is based on packets and can be synchronous or asynchronous. The channel access is handled through Carrier-Sense Multiple Access with Collision Detection (CSMA/CD) [67].

Despite its higher bandwidth compared to the CAN bus, at the time of writing, automotive Ethernet is mostly used for measurement and diagnostics, and seems still far from becoming the backbone of in-vehicle communication. The main challenge for its global-scale adoption is the low robustness to electromagnetic noise and to physical failures [68] compared to CAN.

2.1.7 Benchmarking

Table 2 compares the properties of the in-vehicle networks presented in this section. In particular, the table highlights that the in-vehicle networks are employed for different purposes according to the needs of the OEM in terms of function, cost, fault tolerance and bandwidth. Typically, different buses are embedded in a vehicle to optimise the communication of each of its components, resulting in a hierarchy of networks.

Despite the increasing presence of LIN, FlexRay, MOST and Ethernet in newly released vehicles, the CAN bus is still the only network that can be found on the totality of the cars and

Table 2: Comparison between the main in-vehicle networks.

	CAN	**CAN FD**	**LIN**	**FlexRay**	**MOST**	**Ethernet**
Usage	Powertrain	Powertrain	Body electronics	High performance powertrain safety	Mutimedia	Diagnostics, upgrades
Cost	Medium	Medium	Low	High	Very High	Low
Data rate	1 Mbit/s	5 Mbit/s	20 kbit/s	10 Mbit/s	Up to 150 Mbit/s (MOST150)	Up to 1000 Mbit/s (IEEE 802.3bp)
Latency	Load dependent	Load dependent	Constant	Constant	Data stream	Load dependent, data stream
Physical Layer	2 wire bus	2 wire bus	1 wire bus	2 or 4 wire bus	Dual wire based fiber optic	2 wire bidirectional bus
Network Topology	MultiMaster	MultiMaster	SingleMaster	Multidrop bus, Star, Hybrid	Peer to Peer (MultiMaster)	Star
Message transmission type	Async.	Async.	Sync.	Async. and Sync.	Async., Sync. and Isochronous	Sync. and Async
Access Control	Polling	Polling	CSMA/CA	TDMA, FTDMA	TDMA, CSMA/CA	CSMA/CD
Payload	0–8 Byte	0–64 Byte	2, 4 and 8 Byte	256 Byte	60 Byte	46 Byte
Message identification	Identifier	Identifier	Identifier	Time slot	–	Source + Destination

trucks worldwide. Due to its critical role in the commercial vehicles and its massive market penetration [69], it is the bus on which scientific research has been conducted the most in the field of security. In addition, the CAN data is widely exploited by automotive companies to provide aftermarket services. For these reasons, the focus of this dissertation is the CAN bus and, in particular, the interpretation and usage of the data transiting on it.

2.2 CAN Data

In CAN, the data is not encrypted, but rather encoded according to the proprietary format of the OEM, which is kept secret from the general public. The actual information is contained in the signals, which are chunks of data in the payload that describe the behaviour of a vehicle function or telemetry over time. A signal is defined by the following characteristics:

- **Boundaries** – between one bit for status signals carrying binary information (e.g. the seat belt is fasten or not) and many bits for signals carrying more complex information.
- **Endianness** – is the order in which data is sent over a communication channel. In the Big Endian (BE) format, the bytes are ordered from the most significant to the least significant. By contrast, according to the Little Endian (LE) format, the bytes appear from the least significant to the most significant. In BE, the bits follow the same order as the bytes, i.e., the most significant bit appears first and then the significance decreases until the last bit. As a consequence, this format is immediately interpretable by humans because it reflects our daily way of reading decimal numbers. Vice versa in LE, the significance of the bits does not reflect the significance of the bytes.
- **Semantic meaning** – represents the telemetry/vehicle function that the signal encapsulates (e.g. "vehicle speed", "front left door status", "engine temperature").
- **Signedness** – indicates whether the signal is signed or not, i.e., if the first bit corresponds to the Most Significant Bit (MSB) or to the sign of the signal. Signals that are signed usually refer to telemetries that can have negative values, e.g., the outdoor temperature or the angle of the steering wheel.

- **Format** – typically, to find the actual human-interpretable physical value v carried by a signal s, it is not sufficient to parse the signal encoding from binary (or hexadecimal) to decimal format. Let r be the raw decimal value of the signal, to obtain v a scale factor f must be applied and an offset o must be added, as shown in Equation (1):

  $$v_s = f_s \cdot r_s + o_s \tag{1}$$

  To be noted that factor and offset of a signal can be equal to, respectively, 1 and 0. In such case, the signal is immediately interpretable.

All the presented properties are fixed. Namely, once they are known for a signal, they can be used to correctly interpret the signal at any time. Additionally, frames associated with the same ID traditionally always contain the same signals. Hence, once the position, the meaning, and the format of the signals related to a certain ID are known, its frames can be interpreted at any time.

However, some OEMs have recently started adopting *multiplexing* to attribute different sets of signals to frames with the same ID [70]. In *simple* multiplexing, the value of a reference *multiplexor* signal is used to identify the set of signals that can be found in the same frame. In *extended* multiplexing, there are multiple multiplexors, ordered according to a hierarchy, that define the content of the rest of the frame. Similarly to the standard frames, frames with multiplexed payloads are sent periodically. However, in order to avoid collisions among frames associated with the same ID, each frame is sent according to a delay with respect to the base period of the cycle [71]. Figure 4 illustrates an example of extended multiplexing.

As of today, the totality of work on reverse engineering has focused exclusively on non-multiplex CAN data. For this reason, hereafter, when referring to CAN frames the absence of multiplexing is implicitly assumed. Nonetheless, in Section 12, we discuss how such a technique may impact reverse engineering and we propose a possible solution.

2.3 Taxonomy of CAN signals

According to [23], [27], [28], signals can be grouped into the following categories:

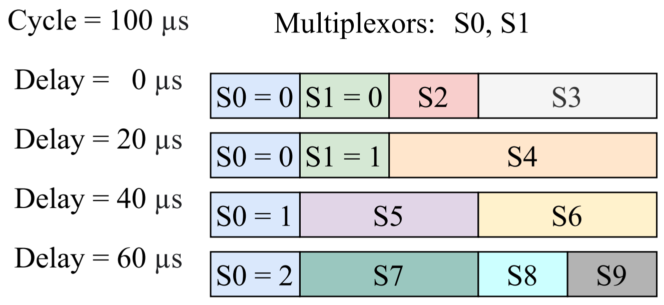


Figure 4: Example of extended multiplexing. S0 and S1 correspond to the multiplexor signals. Each combination of values uniquely identify a subset of signals.

- **Physical** – embed the dynamic nature of a vehicle. They are typically equal or longer than 6 bit and are mostly related to the activities in the powertrain. They carry information about telemetries related to critical real-time events, such as vehicle speed, steering wheel angle, engine temperature etc.
- **Status** – represent a limited set of options, or *states*. Marchetti and Stabili [28] divide the status signals into *multi-value* and *binary*. Multi-value signals are typically between 2–4 bit long and can represent more than two states. An example of multi-value status signal is the wiper speed. By contrast, binary status signals can represent only two options, e.g. on/off or open/closed. Examples of binary signals are the door status and the seat belt status.
- **Counters** – behave like cyclic counters.
- **Checkcodes** – checkcodes, or *checksum*, within the payload that complement the CRC field.
- **Constant/Unused** – consecutive bits that never change their status and that do not encapsulate any vehicle function or telemetry. They are used as buffers between signals.

Figure 5 summarises the taxonomy of CAN signals.

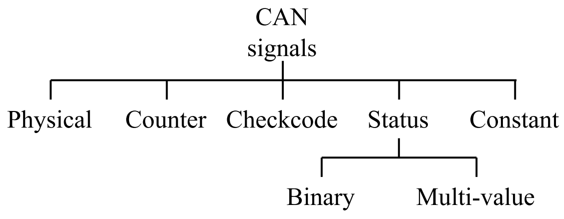


Figure 5: Taxonomy of CAN signals.

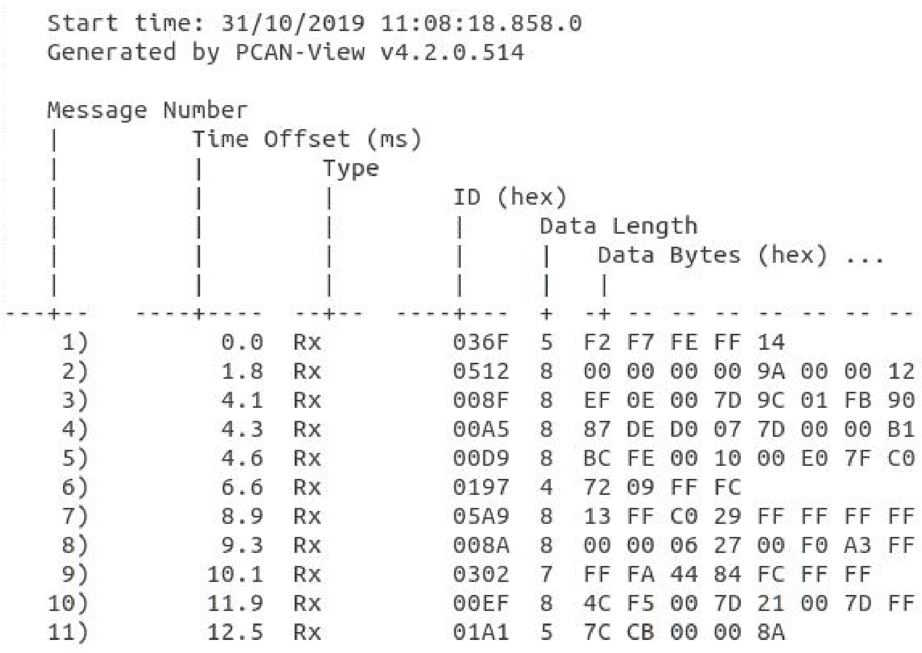
2.4 CAN Data Collection and Diagnostics

The most common way to log the data from the CAN bus is to physically connect a CAN *logger* to the bus. There are several models of CAN loggers commercially available, the most popular being the CLX000 series by CSS Electronics [72], PCAN by PEAK System [48] and Leaf by Kvaser [73], and projects for open-source boards, such as Arduino [74] and Raspberry Pi [75]. In this work, for the data collection a CL2000, a PCAN, as well as a Raspberry Pi 3b with a PiCAN Duo shield [50] are employed.

The CAN loggers are typically equipped with a male 16-pin (2x8) J1962 connector – ISO 15031-3 [76] –, have a timestamp resolution in the order of microseconds, and can automatically detect the bit rate of the CAN bus. In addition, some loggers have a USB or WiFi interface and can store data locally or on a cloud platform. A logger usually outputs a log, or *trace*, as a text file containing the list of received frames, sorted by the timestamp and reporting the DLC, the ID and the payload (expressed in hexadecimal values). Figure 6 provides an example of a CAN trace.

At the time of writing, the most straightforward manner to attach a logger to the CAN bus in most vehicles is through the OBD-II port. The OBD-II port is based on a female 16-pin (2x8) J1962 connector, and it is commonly located within 0.60 m from the steering wheel [77]. Released by SAE International (formerly known as Society of Automotive Engineers (SAE)), nowadays it is present in most modern gas powered vehicles worldwide as it was enforced in the US and EU, respectively, in 1996 and 2001, and later in a number of other countries. An example of OBD-II port is shown in Figure 7.

The purpose of OBD-II is to request emissions related data from a vehicle by sending CAN



```
Start time: 31/10/2019 11:08:18.858.0
Generated by PCAN-View v4.2.0.514

Message Number
|         Time Offset (ms)
|         |        Type
|         |        |        ID (hex)
|         |        |        |     Data Length
|         |        |        |     |   Data Bytes (hex) ...
|         |        |        |     |   |
---+--   ----+----  --+--  ----+---  +  -+ -- -- -- -- -- -- --
   1)         0.0  Rx         036F  5  F2 F7 FE FF 14
   2)         1.8  Rx         0512  8  00 00 00 00 9A 00 00 12
   3)         4.1  Rx         008F  8  EF 0E 00 7D 9C 01 FB 90
   4)         4.3  Rx         00A5  8  87 DE D0 07 7D 00 00 B1
   5)         4.6  Rx         00D9  8  BC FE 00 10 00 E0 7F C0
   6)         6.6  Rx         0197  4  72 09 FF FC
   7)         8.9  Rx         05A9  8  13 FF C0 29 FF FF FF FF
   8)         9.3  Rx         008A  8  00 00 06 27 00 F0 A3 FF
   9)        10.1  Rx         0302  7  FF FA 44 84 FC FF FF
  10)        11.9  Rx         00EF  8  4C F5 00 7D 21 00 7D FF
  11)        12.5  Rx         01A1  5  7C CB 00 00 8A
```

Figure 6: Example of a CAN log extracted with a PCAN from a BMW X1, model year 2015.

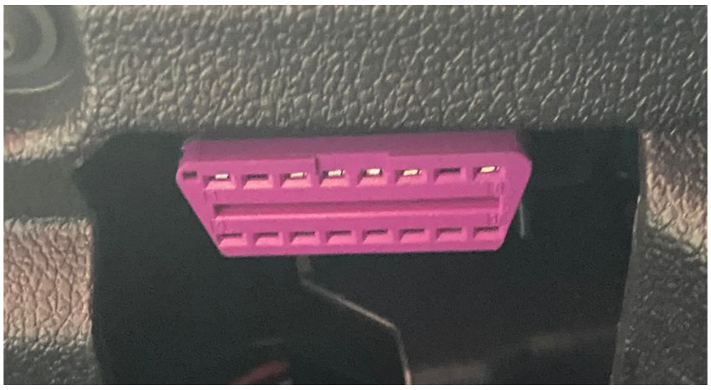


Figure 7: OBD-II port in a Cupra Formentor, model year 2021.

messages, identified by OBD-II PIDs [78]. It covers the two layers of the OSI model on which CAN is based plus the network layer. Differently from the CAN IDs, the OBD-II Parameters IDs (PIDs) are defined by the OBD-II protocol and the same for all the vehicles equipped with the OBD-II.

While its main purpose is the request of diagnostic information related to the emissions, the OBD-II port typically allows the logging of standard CAN frames as well. In the recent years, however, a number of OEMs have started placing a firewall between the OBD-II port and the CAN bus, thus preventing the logging of non-diagnostic CAN data. To extract data from vehicle models having such a configuration is necessary to identify the CAN wires and use an insulation piercing clip or connector with an adapter for the CAN logger.

Another widely adopted diagnostics communication protocol is Unified Diagnostic Services (UDS) – ISO 14229-1 [79]. UDS is the direct descendant of Keyword Protocol (KWP)2000 – ISO 14230-3 [80] – and Diagnostic communication over Controller Area Network (DoCAN) – ISO 15765-3 [81]. This protocol is adopted by the majority of Tier 1 suppliers and, therefore, implemented on a large number of ECUs.

UDS covers the fifth and seventh layers of the OSI model (respectively, session and application). ECUs with UDS services enabled can be interrogated by diagnostic tools connected to the bus for the detection of faults. The parameters of the requests are encapsulated in the payload of CAN frames, along with their unique Service ID (SID). Once faults are detected, the firmware of the defected ECUs can be *flashed*, i.e. uploaded. UDS can be employed not only for the detection of faults, but also for their prevention. In fact, through the usage of special functions called *routines*, it is possible to understand and predict the usual behaviour of the ECUs over time.

While OBD-II and UDS are both employed for the diagnostics, their purpose is different. OBD-II targets specific ECUs and vehicle functions – those related to the emissions, e.g. oxygen sensors, engine control unit etc. –, and it is *on-board*, i.e. it is integrated in the vehicle and can be employed by both technicians and users. By contrast, UDS can be used and extended for any type of diagnostics and it is *off-board*, i.e. the diagnostic analysis is performed when the vehicle is stationary in the service garage. Furthermore, due to its versatility, UDS

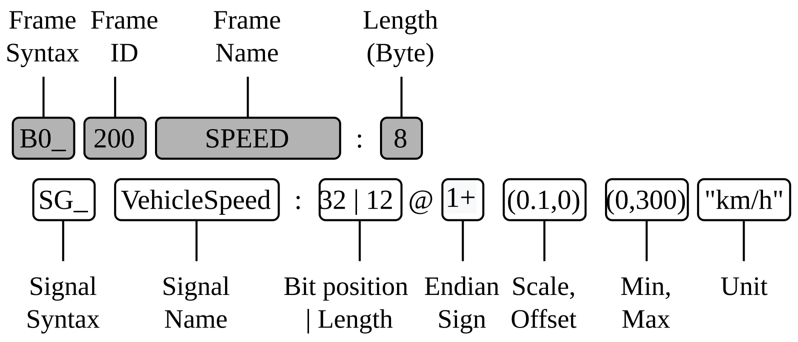


Figure 8: An example of format description in a DBC file.

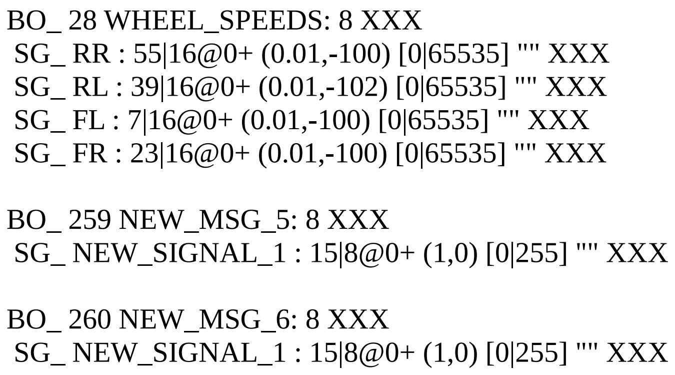
has been incorporated in a number of standards, the most notable being AUTOSAR [82].

Both OBD-II and UDS protocols have been employed for CAN bus reverse engineering [23]–[26], [30], [31], [83]. However, their usage is still limited, since many CAN standard frames are out of their sphere of activity. The original work presented in this dissertation does not make use of either OBD-II and UDS diagnostic frames, but rather focuses on the reverse engineering of the standard frames transiting on the CAN bus.

2.5 DBC Files

To evaluate a CAN bus reverse engineering method on a target vehicle, apart from its CAN trace, it is necessary to have the ground truth to validate the results against. The standard to store such information is Database CAN (DBC), released by Vector Informatik [84]. A DBC file stores the characteristics described in Section 2.2 and the maximum and minimum values that a signal can assume. Figure 8 illustrates an example of format description in a DBC file. The top part of the figure, highlighted in grey, shows the syntax of a frame, while the lower part represents an example of a signal within the frame.

Typically, the *original* DBC files owned by OEMs are not accessible by the general public. An alternative option for obtaining ground truth datasets for validation is to employ DBC files obtained by third-party expert technicians through manual reverse engineering. These DBC files can be defined as *generated*. The disadvantage of generated DBC files is that, generally, they do not cover all the information related to the target vehicle, as some signals may have



```
BO_ 28 WHEEL_SPEEDS: 8 XXX
 SG_ RR : 55|16@0+ (0.01,-100) [0|65535] "" XXX
 SG_ RL : 39|16@0+ (0.01,-102) [0|65535] "" XXX
 SG_ FL : 7|16@0+ (0.01,-100) [0|65535] "" XXX
 SG_ FR : 23|16@0+ (0.01,-100) [0|65535] "" XXX

BO_ 259 NEW_MSG_5: 8 XXX
 SG_ NEW_SIGNAL_1 : 15|8@0+ (1,0) [0|255] "" XXX

BO_ 260 NEW_MSG_6: 8 XXX
 SG_ NEW_SIGNAL_1 : 15|8@0+ (1,0) [0|255] "" XXX
```

Figure 9: An example of wheel speed signals – front left (FL), front right (FR), rear left(RL) and rear right (RR) – from a generated DBC (OpenDBC) extracted from a Mazda 3, model year 2019. The generated DBC file is incomplete as the signals in the following two frames were not fully translated.

not been spotted or fully decoded. Figure 9 is an example of signals in a generated file in the Open DBC repository by Comma AI [52]. Is it to be noted that the four wheel speeds have been almost fully reverse engineered except for the unit, while for others only the boundaries have been identified, with no information related to their semantic meaning (and, supposedly, scale factor and offset too).

In this work, the validation of the results is based on two datasets of generated DBC:

- Set of Derived DBC (SDDBC) – a set of 477 generated DBC files related to vehicle models from 28 different makers from EU, USA, Japan and South Korea. This dataset was generated by our partner company, Xee [51], through a process of manual reverse engineering, starting from 2016. With an aggregated number of 21 526 decoded signals related to 477 vehicles, to the best of our knowledge this is the most extensive and diverse set of generated DBC files commercially available.

- OpenDBC, a set of generated DBC files publicly available online [52]. Differently to SDDBC, not all signals present in OpenDBC files have been fully decoded. For some signals, only the location is identified, with no information concerning semantic meaning nor the format. For the sake of the validation, we have preliminarily discarded these signals from the ground truth.

3

Introduction to CAN Bus Reverse Engineering

The objective of CAN reverse engineering is to locate the position of signals within frames and *translate* their semantic meaning and format.

In the first part of this section we introduce the field of reverse engineering and we contextualise our work accordingly. In the second part, we compare methodologies and results obtained in the field of automated CAN bus reverse engineering. We evaluate the work proposed in literature about tokenisation and translation separately. In particular, we divide the related work according to the underlying approach employed.

Verma *et al.* [24] are the first to categorise methodologies on CAN reverse engineering based on signals properties that they can decode. However, their categorisation is incomplete and unstructured (i.e., the authors do not highlight commonalities and differences of the presented methodologies). As a consequence, this dissertation provides the first comprehensive overview of the work proposed and the related results achieved in the field of CAN reverse engineering.

3.1 Reverse Engineering

Reverse engineering is a deductive process aiming at identifying the components of a target device, system, software, or process, and understanding how they interact with each other.

The ultimate goal is to create an abstract representation of the target thanks to the newly acquired knowledge [85].

The history of reverse engineering is tied with the evolution of war technology. In 260 BC, the Romans used a shipwrecked Carthaginian Quinquereme as a blueprint for the production of their new fleet of 120 warships, which largely contributed to their victory in the First Punic War [86]. During WWII, four Boeing B-29 Superfortress bombers were dismantled and reverse engineered in a large-scale operation, after making an emergency landing in the USSR. They were effectively used for the design and production of the Tupolev Tu-4 bomber [87].

Reverse engineering techniques are used in a variety of fields, such as computer, electronic and mechanical engineering, design, and systems biology, as well as for a multitude of research and commercial purposes, such as forensic investigations and competitor analysis [88], [89]. Given the multitude of targets and their intrinsically different nature, the techniques presented in the literature vary greatly. Nonetheless, it is possible to identify three general steps that are typically followed in the process:

1. **Information extraction** – The target is analysed and its constituent components are identified.

2. **Modelling** – The newly acquired information is conceptualised, i.e. an abstract model representing the whole structure is designed.

3. **Validation** – The abstract model is tested in a multitude of scenarios to validate the findings.

A field in which reverse engineering has gained particular popularity is software engineering. Often it happens that the information regarding the design of a software is lost over time and/or the software has become obsolete and, thus, incompatible with modern systems. In this case, reverse engineering is also known as *design recovery* and it is a a helpful methodology to speed up the understanding of the source code and/or its re-purposing [90].

Reverse engineering can be employed not only on an end product, but at any stage of a software development. Its output is useful to take informed decisions during the software development by, for instance, offering graphical representation of the code structure. This process is known as *re-documentation* [91]. A number of Universal Modeling Language (UML) tools for the analysis of source code can be seen as reverse engineering processes for re-documentation [92], [93]. Compared to the direct analysis of the source code, re-documentation provides a more structured and high level understanding of the software. For this reason, it is employed for identifying flaws in the system design, as well as bugs and vulnerabilities.

Reverse engineering is also known for being widely adopted in illegal activities [94]. Similarly to software developers, black hat hackers typically exploit reverse engineering techniques to detect vulnerabilities in operating systems or software and steal information or inject malware [95]. Another notable example is the removal of the copyright protection of the source code or media, also known as *cracking.* Vice versa, reverse engineering techniques can be employed to discover unauthorised copies of the source code.

When the source code is not available, reverse engineering is performed on the machine code or bytecode and can be referred to as *reverse code engineering* [95]. This can be grouped into the following categories: *decompilation*, *disassembly*, and *protocol reverse engineering.*

Decompilation makes use of *decompilers* that generate source code in a high-level language from the machine code of executable programs. This methodology typically produces imperfect results, leaving part of the code obfuscated. The success rate of decompilers largely depends on the availability of debug data and metadata, e.g. produced by virtual machines. Examples of widely used decompilers are JAD for Java [96] and Reflector for .NET [97].

Disassembly is based on *disassemblers*, which are capable of translating binary code into assembly language. Being the main purpose of a disassembler to provide insights for humans, the generated assembly code is generally formatted with the aim of human readability rather than an input for assemblers. A notable disassembler is IDA Pro [98].

Given a networking protocol whose formal specifications are unknown, the goal of protocol reverse engineering is to infer its parameters, formats, syntax and semantics [99]. It can be achieved following two distinct approaches, (i) study the dynamic behaviour of an application sending messages according to the protocol, and (ii) sniff and analyse the traffic exchanged between two or multiple hosts.

In the first approach, taint checking is employed to locate the code that parses the messages and correlate it with the messages. The output is called *execution trace* and it is used to identify the fields and content type of messages. The main limitation of this approach is its high dependency on the type of system and programming language, which makes it difficult to generalise. Moreover, similarly to the decompilation, this approach is oftentimes limited by privacy prevention techniques.

The second approach is based on logging the traffic transiting a network into a *network trace*, and identifying patterns and correlations between packets and messages in it. The decoding process is mostly composed of two steps: *syntax inference* and *semantic inference*. Inferring the syntax means discovering the field boundaries, also known as *tokenisation*, as well as the endianness (see Section 2.2) and format rules used to represent the information in the protocol. The semantic inference is the process of understanding the semantic meaning of the data. Network trace based protocol reverse engineering can be performed with the help of packet analyser tools, such as Wireshark [100], which allow inspecting the data manually. Research has focused on automating this process with the aim of improving the performance and allow to deal with complex protocols. The most successful approaches presented in literature are based on Finite State Machine (FSM) automata and ML algorithms. The former are used to map sequences of messages and their relations [101]. The latter are employed to find underlying patterns and similarities between different messages or chunks of data [102], [103].

Automated CAN reverse engineering is a process whose goal is to identify the semantic meaning and format of signals contained in data payload by sniffing and analysing traffic traces generated by ECUs connected to the CAN bus. It is a protocol network trace based task and, as such, it shares most of the characteristics of this process, including tokenisation, and often implies the usage of ML techniques. However, due to its specific characteristics, the methodology for CAN reverse engineering can also consistently diverge from other practices presented in literature for protocol network based reverse engineering (see Section 3). Figure 10 highlights the location of CAN reverse engineering with respect to the taxonomy of reverse engineering tasks.

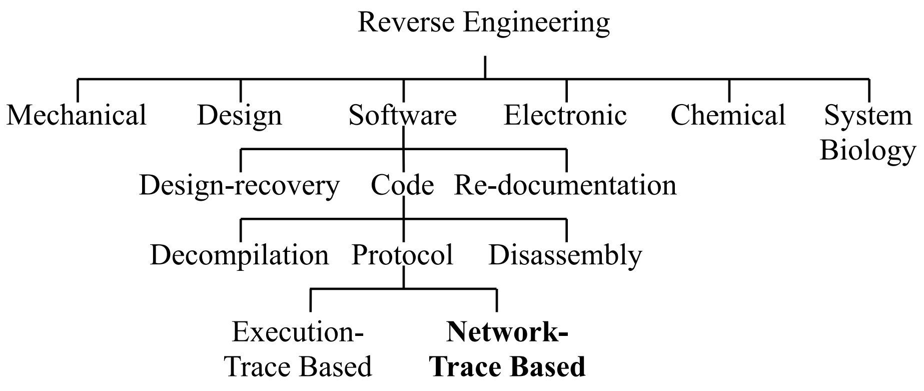


Figure 10: Taxonomy of Reverse Engineering. The reverse engineering of CAN bus is a network trace-based protocol task.

3.2 From manual CAN reverse engineering to full automation

As for the majority of the reverse engineering tasks, the decoding of the CAN bus has been initially executed manually. The manual approach is performed by a trained human operator, who triggers events in the vehicle and analyses the changes in the CAN traffic [104], [105]. This process involves the activation and deactivation of sensors (e.g., the wipers are actuated, the car is driven at certain speed), and/or the injection of PIDs through the OBD-II port to generate a response from the network. The impact that such operations have on the data transiting on the CAN bus is monitored in real time with the help of specialised tools, such as Wireshark, CANtrace, and CANalyzer. These tools allow filtering the CAN data according to selected frame IDs, thus providing a better understanding of the dynamics of semantically consecutive information. Moreover, they can provide additional useful information regarding the bus load, counters for errors, etc.

Manual CAN bus reverse engineering has proven to be a reliable tool for obtaining a clear understanding of the CAN data. However, it can take from hours to days and it is not scalable, i.e. the reverse engineering of a new target vehicle requires a similar number of operations and comparable time. In the recent years, the automation of this process has captured the attention of the scientific community and industrial players, who aim for standardised, fast and scalable solutions.

Table 3: Categorisation of CAN reverse engineering into manual, semi- and fully-automated.

	Manual	**Semi-Automated**	**Fully-Automated**
Human expertise	High	Low	None
Context	Dependent	Dependent	Partially or fully independent
Data Collection Time	Hours to days	Minutes to hours	Seconds to minutes
Decoding time	Hours to days	Minutes	Minutes
Scalability	Low	Medium	High

Table 3 summarises the characteristics of manual, semi- and fully-automated CAN bus reverse engineering. The table shows that the main difference between the three approaches is the level of human expertise and effort required. In manual reverse engineering, an expert human operator with a solid background knowledge of the CAN protocol is a key asset to achieve meaningful results. In this case, the operator must be skilled in the usage of the analyser tools and trained in interpreting the changes in the traffic. In the semi-automated approach, the operator does not need to be educated on CAN, but must be instructed to follow specific procedures to generate the events in the vehicle. Full automation, instead, does not require any human expertise related to reverse engineering or CAN. Indeed, an individual inside the vehicle can be potentially unaware that this process is taking place. This is an ideal scenario for companies who have remote access to their clients' vehicles – i.e., through the installation of proprietary sensors – and can collect CAN data at any time and process it offline.

Another relevant difference between these categories is the need for ground information regarding the data collection, in terms of driver profile, operations performed within the vehicle, road and traffic conditions. In the manual and semi-automated approach, it is fundamental to understand the correlations between the data collection context and the CAN traffic. Ground truth data can be collected with the usage of additional hardware (e.g., GPS, or IMU sensors) by inputting information manually (i.e., what is the recorded engine revolutions per minute (RPM) as shown by the dashboard) or by injecting diagnostic messages. The goal of full automation is to minimise the dependency from this kind of information and enable reverse engineering regardless of the data collection conditions.

Related work on semi-automated and automated reverse engineering has shown that the data collection time can be reduced by, respectively, one and two orders of magnitude compared to the manual approach [23], [24]. For what concerns the decoding time in the semi-automated and fully-automated approaches, it refers to the execution time of the software. Compared to the manual methodology, it is also reduced by two orders of magnitude [23], [24], [106]. However, it is to be noted that, since the manual decoding is performed in real time during the data collection, the time for the two operations overlaps and, therefore, should not be summed up.

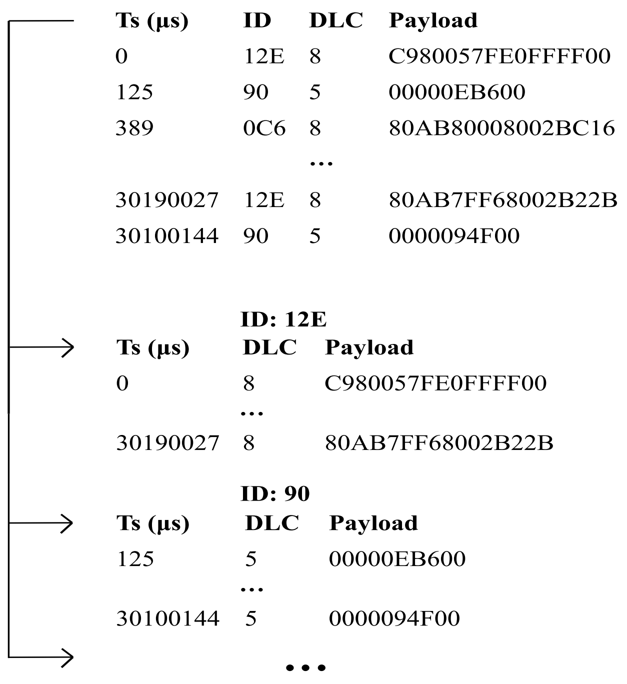
Finally, another difference among reverse engineering methodologies concerns their scalability. In the manual approach, the same or similar manual operations have to be performed by the human operator for every target vehicle, which makes the process poorly scalable. In the semi- and fully-automated approach, the more vehicles are processed, the more accurate the reverse engineering of the new target vehicle typically becomes (i.e., by training ML to recognise a larger number of signals [31]).

3.3 Tokenisation

The goal of tokenisation is to identify the sequence of bits that correspond to each signal within the payload of the frames. The resulting output, the *tokens*, can be described as signals whose location in the payload is known but whose semantic meaning and format have yet to be translated.

The first step, common to all tokenisation approaches, is the decomposition of the CAN trace in sub-traces, one for each ID. Each sub-trace is ordered according to the timestamp, and corresponds to the time series of frames associated to the same ID which, thus, carry semantically consecutive data. Figure 11 illustrates the process of decomposing a CAN trace into multiple sub-traces.

After this step, related work proceeds following one of two approaches: (i) extract the set of tokens that maximises a score function among a set of possible tokens combinations, or (ii) compute the Bit Flip Rates (BFRs) array from the payload time series, and scan it to identify boundaries based on drops in the value.



Ts (µs)	ID	DLC	Payload
0	12E	8	C980057FE0FFFF00
125	90	5	00000EB600
389	0C6	8	80AB80008002BC16
		...	
30190027	12E	8	80AB7FF68002B22B
30100144	90	5	0000094F00

ID: 12E

Ts (µs)	DLC	Payload
0	8	C980057FE0FFFF00
	...	
30190027	8	80AB7FF68002B22B

ID: 90

Ts (µs)	DLC	Payload
125	5	00000EB600
	...	
30100144	5	0000094F00

• • •

Figure 11: Decomposition of a CAN trace into N sub-traces.

3.3.1 Tokenisation based on combinatorial optimisation

While developing an Anomaly Detection (AD) system, Markovitz and Wool [27] were the first to design an algorithm for tokenisation, They were also the first to identify different categories of signals: physical, constant/unused, multi-values, and counters. The authors propose an approach based on a 64x64 triangular matrix, which is employed to evaluate all possible 2080 tokens within the frame payload (the authors take into consideration only payloads with the maximum length of 8 Byte). For each candidate token, the number of distinct values assumed in the trace is calculated. Constant/unused portions of the trace are identified by the fact that they assume only one value throughout the trace, while multi-values are characterised by a limited number of values. According to the authors, physical signals and counters are signals with high *density*, i.e. a large number of the values theoretically available are displayed by the signal during the trace. A score is attributed to each candidate signal based on the specific characteristics of its category. Through a greedy algorithm, the authors identify the final set of non-overlapping signals based on their score.

Verma *et al.* [30] present Automatic CAN Tokenization and Translation (ACTT), an algorithm that performs tokenisation and translation of the signals simultaneously. The method initially identifies the portions of the target trace triggered by PID requests injected through the OBD-II port during the driving session in the target vehicle and obtains labelled times series. Then, the constant bits are extracted for each ID and all the possible combinations of tokens are computed on the rest of the bits. A fitness score is calculated between the time series of each token in the set and the Diagnostic IDs (DIDs) times series through a linear regression. Finally, a dynamic programming-based scheduling algorithm outputs the set of non-overlapping tokens which maximises the overall fitness score. The authors try to address the endianness but, erroneously consider the bit order instead of the byte-order, which truly defines the endianness in CAN bus.

In 2020, Frassinelli *et al.* [53] equip their novel tool, AutoCAN, with a tokeniser based on a greedy approach. For each ID, the tokeniser starts from the first bit of the payload and iteratively adds bits to define the other boundary of the token. At every iteration, the algorithm evaluates the *plausibility* of the considered token by verifying that its properties match those of any of the signal categories (physical, counter, checksum, status). Let us consider a token starting at position i, and whose currently investigated end bit is at position j. If the token (i, j) is plausible, a new bit is added and a new plausibility check is performed. If not, a boundary is identified in position $j-1$, and j is the start bit of the following token. The authors also refer to the little-endianness. However, they do not specify how to distinguish little-endian signals from big endian signals, which are not mentioned in the paper. The results show a correctly extracted tokens (CE) rate higher than 90 %.

In their tool, CAN-D, Verma *et al.* [24] propose an approach that considers the conditional probability of each bit b_i flipping, and the difference between the conditional bit flip probabilities of the following two bits. Once the probability of each bit in the payload of being a signal boundary is calculated, the algorithm extracts the set of all possible valid tokens. This set includes also tokens in little-endian format, which have been extracted independently from each other. The authors evaluate the possibility that a single payload can contain tokens encoded according to different endianness. In fact, they observe that, while no case of signals within the same payload but with different endianness has been recorded, the CAN protocol does not indicate constraints regarding this possibility. An optimisation algorithm aims to find the optimal balance between partitioning the payload into too many tokens and allocating too many bits to the same token. This algorithm can provide distinct outputs. In this case, the solution that includes the highest number of tokens overall and the highest number of big endianness tokens is chosen as optimal. Both approaches were tested on 10 vehicles from different manufacturers and mostly validated against generated DBC files provided by OpenDBC. The results show an average F-Score, accuracy, and precision around or superior to 90 %.

3.3.2 Tokenisation based on BFR

In concurrent works, Marchetti and Stabili [28] and Nolan *et al.* [107] introduce, respectively, READ and Transition Aggregated N-Grams (TANG), which are tokenisation algorithms based on the BFR.

For each ID time series, a *BFR array* $B = [b_0, b_1, ..., b_n]$ is calculated, where n is the length of the frame payload. Each element b_i corresponds to the BFR of the i^{th} bit throughout the time series. A bit flip corresponds to a change in the status of a bit in a consecutive payloads, from 0 to 1 or vice versa. The total Bit Flip Count (BFC) is the number of times that a bit flips throughout the time series. Let F be the number of consecutive payloads that constitute the time series, BFR is calculated as BFC/(F - 1). The BFR array is then scanned from left to right looking for a significant drop in the BFR between a b_i and b_{i+1}. When such a decrease in the BFR occurs, a boundary between two signals is found.

The underlying logic behind this approach is that the Less Significant Bits of physical signals usually flip more than the Most Significant Bits, due to the deterministic and continuous nature of the phenomena that these signals represent. As a consequence, when a consistent decrease in the value is found between b_i and b_{i+1}, b_i is likely to be the Less Significant Bit of the current signal, while b_{i+1} the Most Significant Bit of the next signal. A difference between READ and TANG is the factor defining the minimum decrease in the bit flip to identify a boundary: 2 for TANG and 10 for READ. Figure 12 shows an example of tokenisation based

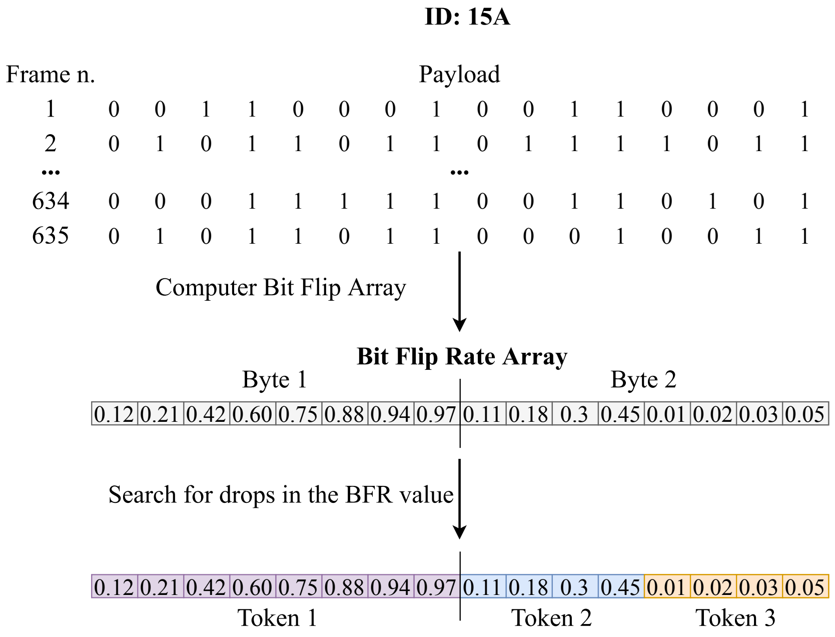


Figure 12: Example of tokenisation of the ID 15A. The BFR is calculated on the payloads of consecutive frames associated to the ID. The array is scanned from left to the right to look for drops in the BFR values. Through this technique, 3 tokens are found.

on the BFR.

READ has been further improved by Pesé *et al.* [23]. Specifically, instead of defining a binary threshold to determine the presence of a boundary between two bits, the authors express the decrease in two consecute BFR values as a percentage. The overall probabilities of the bits of being a boundary are then taken into account to determine the final set of tokens. In their work [23], the authors also define three metrics: CE, total number of signals in the DBC file (TDBC) and total extracted signals (TE). Then, they evaluate their tokenisation algorithm according to the ratios CE/TE and TE/TDBC.

Choi *et al.* [29] challenge the assumptions made by Marchetti and Stabili [28] regarding the BFR of checkcodes and counters. Regarding the checkcodes, they observe that not always the BFR values are distributed around 0.5. Their algorithm starts from a reference BFR of 0.3 and continuously check whether this property is met throughout the trace. For what concerns the counters, the algorithm simply observes whether the overall value of the counter has increased by 1 at every new frame sending. The authors present a novel methodology to identify physical signals as well, which does not calculate the BFR array on the whole payload time series. Following a sliding window approach, the algorithm generates multiple sub-time series, or *windows*, and calculates a BFR array for each of them. Then, each bit is evaluated independently. The BFR related to the windows are put in one time series for each single bit and a correlation with the following bit is searched. When correlations are found between two or multiple consecutive bits, they are labelled as belonging to the same signal.

The main limitation of this work is the assumption that the observed value of the counter increases by exactly 1 with each frame sending. Counters related to physical phenomena, such as the fuel consumption counter, can speed up or slow down according to the progression of the telemetry taken into consideration. As a consequence, while displaying an evident cyclic behaviour, i.e., a monotonic increase in the values followed by a reset, their value can change by more than 1 between one frame sending and the following.

3.4 Translation

The goal of the translation is to disclose the signedness, offset, scale factor, and unit (in case of physical signals) of the found tokens. In this section, we review the state-of-the-art techniques that achieve semi-automated and fully automated translation. We divide the work on translation according to the methodology adopted: (i) based on PIDs, (ii) based on ML, (iii) based on semantic taxonomy, (iv) based on companion apps.

To be noted that a significant number of the methodologies presented in this survey are based on more than one approach, as illustrated in Figure 13. However, for the sake of readability, in this section we present each work in the category that describes its most distinctive characteristics.

3.4.1 Translation based on PIDs

The algorithms belonging to this class are based on injecting PIDs through the OBD-II port and observing the subsequent changes in the traffic.

Huybrechts *et al.* [31] were the first to exploit OBD-II PIDs in the scope of CAN reverse engineering. They log CAN data while driving in a dynamic context and injecting diagnostics

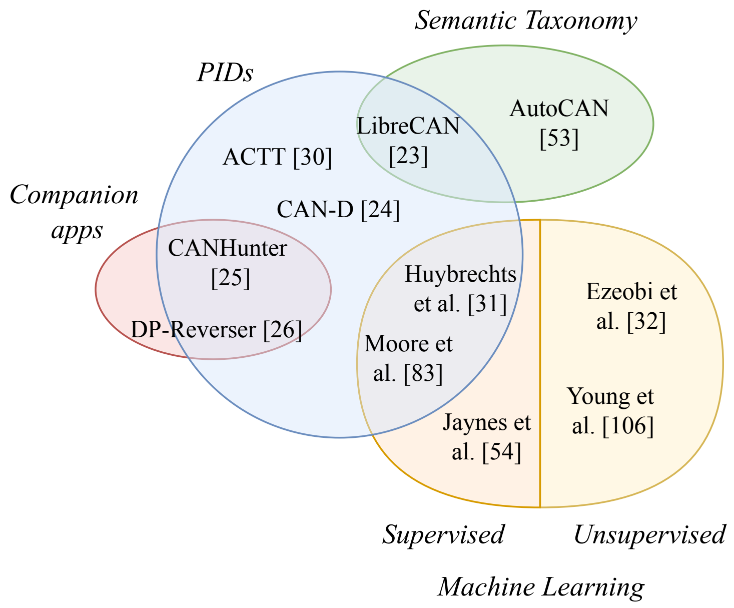


Figure 13: Classification of related work on CAN bus reverse engineering.

requests. Then, the tool compares each byte (or combination of bytes) of the collected traffic with the triggered diagnostic signals and calculates the similarity between them. The semantic meaning of the chunk of traffic scoring the highest similarity is then identified. The tool is tested on two gas-fuelled vehicles and validated against a generated ground truth, managing to map around 10 % of the traffic in the two vehicles.

In their novel tool, CAN-D, Verma *et al.* [24] are the first to introduce a translation method capable of decoding the signedness of the signals. The signedness is decoded based on the assumption that if a token is signed, both of its Most Significant Bits flip from 0 to 1 when it goes from positive to negative values, and from 1 to 0 vice versa. The logic behind this approach is that signed signals have continuity in the value of two MSBs while, in the case of unsigned signals, the flipping of such bits show changes in the value of the signal unlikely to represent telemetries in the real world. The translation of the signals' format is a continuation of the work in [30] (see Section 3.3) and consists of finding linear correlations between the extracted tokens and diagnostic messages. Based on a testing set composed of 10 vehicles, whose ground truth is available in CommaAI, CAN-D achieves a reduction of 20% in the l^1

error between the translated signals and the ground truth compared to LibreCAN.

The injection of PIDs is a useful approach that can be employed to accurately translate a number of signals critical for the functioning of the vehicle. In particular, since the formats of the responding signals are publicly available, once the semantic of a signal has been correctly identified, its format becomes also known, and it can be used for the identification of other semantically related signals. The main limitation of this approach is that not all the signals can be triggered through diagnostic requests, in particular those not related to the powertrain.

3.4.2 Translation based on ML

ML is used in the scope of CAN reverse engineering to identify the semantic of signals (supervised learning) or infer correlations between signals (unsupervised learning).

Jaynes *et al.* [54] were the first to attempt fully automated CAN bus reverse engineering. They propose a ML approach based on three features – ID, sending frequency, and payload – to identify the semantic meaning of the frames, i.e., what is the vehicle function that they carry. After training a variety of ML classifiers on frames related to five functions in nine vehicles, the authors achieve a precision and recall higher than 80 % with a NN classifier against a generated ground truth. However, as indicated in [24], since this work treats the frame payloads as a whole, this approach is unfit for the translation of signals within frames.

In their paper, Huybrechts *et al.* [31] also propose an alternative approach to the injection of PIDs, which makes use of Long Short-Term Memory (LSTM), a type of neural networks particularly suited for time series classification. The LSTM is trained to identify the semantic meaning of signals in a target vehicle, based on their experience gained on CAN signals of known vehicles. The training of the LSTM is also enriched by GPS data. This second approach scores an accuracy of up to 90 %. The main limitation of both approaches is the absence of tokenisation. The authors wrongly assume a coincidence between signals and payload bytes, which does not make it suitable for signals that are shorter than one byte and/or whose boundaries are not the Most Significant Bit and Less Significant Bit of the bytes. Moreover, having trained the LSTM on only two vehicles it is difficult to assess whether the second approach is generalisable and would scale with a higher number of samples.

Moore *et al.* [83] aim to provide a full mapping of the time relations between the speed-related signals and distinguish the different states of the vehicle in a semi-supervised fashion. The authors initially exploit PIDs to identify the speed-related signals. Subsequently, they employ a combination of Hidden Markov Model (HMM) and Convolutional Neural Network (CNN) to understand the causality between the actions of the driver and the observed data. The authors explore four states of the vehicle speed: acceleration, deceleration, maintain (the speed is constant) and idle (the vehicle is parked). The results show an average accuracy in the identification of the states higher than 90%.

In concurrent works, Ezeobi *et al.* [32] and Young *et al.* [106] adopt unsupervised ML techniques in the field of CAN reverse engineering. In [32], the authors test several clustering techniques, such as Agglomerative Hierarchical Clustering (AHC) [108] and DBSCAN [109], to determine sets of frames which are semantically related. They evaluate their approach against a set of four vehicle models belonging to the same OEM. For validation, the authors employ generated DBC from OpenDBC. AHC identifies four distinct clusters related to an equal number of vehicle functions, achieving a Fowlkes-Mallows score (FM-Score) [110] of over 70 %. To be noted that this approach provides the semantic meaning of only a limited number of signals.

In [106], instead, the authors perform a series of operations within the vehicle at data collection time and manually label the CAN traffic in different time spans accordingly. The CAN log is split in sub-traces, one for each ID, and the time-status labels are used to find correlation between the time series of an ID and a function of the vehicle. Then, an AHC algorithm is used to identify the similarity degree between different frame series. Following this approach, the authors manage to identify eight frames containing five distinct gas-fuelled vehicle functions. Similarly to [54], this approach is limited, in the sense that the extracted information, while offering an high-level understanding of the frames, does not address the individual signals.

ML-based algorithms are useful to identify and/or group signals based on their semantics in a completely automated manner and, thus, provide a clear overview of the dependencies in the data traffic. The main limitation of tools based on supervised learning is that the translation performance depends on the quality of the training set fed to the classifier which, in the case of CAN bus reverse engineering, is largely influenced by the driving context of the collected traces. By contrast, unsupervised learning can be employed in any circumstance, but needs additional ground truth about some of the examined signals in order to correctly interpret the output.

3.4.3 Translation based on semantic taxonomy

This approach consists in exploiting the intrinsic properties of groups of semantically similar signals and the relations that link these groups. A deep understanding of the nature of signals is required, as well as the capability of transposing this knowledge into an software representation. Translation based on the taxonomy of signals requires operations performed in the vehicle that reflect the nature of the searched signals. A human operator is in charge of conducting a precise list of actions in order to complete the data collection. Hence, all work based on the taxonomy of signals is semi-automated.

A first attempt to provide a complete CAN reverse engineering pipeline has been made by Pesé *et al.* [23], who presented LibreCAN, an algorithm composed of three phases. For the data collection, the authors employ a commercial mobile app, installed in the vehicle and duly aligned, to capture IMU and GPS data related to the driving session. During the driving session, they inject OBD-II PIDs to trigger the responses in the bus. Additionally, another trace is collected, where a human operator is instructed through an app to perform specific procedures in the vehicle in order to activate body status signals, e.g. door locks, seat belts etc. A reference trace, where no action is performed in the vehicle is also logged.

The semantic translation of the signals is performed by searching for correspondences between the extracted tokens, the PIDs and the IMU/GPS data, through *normalised cross-correlation*. Once the semantic meaning of the signals is unveiled, LibreCAN performs a *linear regression* between the raw values of the signals and the ground truth data to decode the scale factor and offset. Finally, LibreCAN analyses the difference between the reference trace and the trace related to the body in order to spot the status signals.

LibreCAN is evaluated on four commercial vehicles and validated against the original DBC files provided by the manufacturer. The tests achieve a precision of 82 % in the identification of powertrain signals (phase 2), and an accuracy of around 95 % in the decoding of body signals (phase 3). Furthermore, the authors investigate how the length of the CAN log influences the performance of the algorithm. The results show that driving a vehicle for 30 min in a dynamic context is sufficient to achieve optimal reverse engineering of the powertrain signals. Moreover, the human operator takes 10 min on average to perform all the due operations for the body signals related trace.

Frassinelli *et al.* [53] present AutoCAN, an approach based on finding a chain of correlations between tokens. A small set of tokens (extracted with the methodology presented in Section 3.3), is initially decoded by finding correlations with ground truth data related to the current session, captured through external sensors. Then, a number of mathematical formulas is iteratively extracted through Abstract Syntax Tree (AST) and applied to transform the rest of the tokens time series and find correlations between the output of each of these transformation and the known signals. The underlying rationale is that signals represent telemetries and events in the real world, which are subject to the laws of physics. Hence, it is possible to infer the relations between two tokens by testing all the possible combinations of mathematical properties that link different measurements in a physical system. For example, the token related to the acceleration can be identified by calculating the integral of each token and finding the most convincing correlation with the vehicle speed (in this case, calculated through the Pearson coefficient). AutoCAN is tested on three Electric Vehicles (EVs) and one fuel-propelled vehicle, and validated against generated DBC files obtained from various sources.

The main limitation of this work is the assumption that all the reference signals, through which it is possible to decode the rest of the traffic, can be always easily identified. While this is a reasonable assumption for what concerns most of the powertrain signals (e.g. using the GPS to preliminarily identify the vehicle speed), it is not necessary true for a number of other signals, such as those concerning the battery voltage. Finally, the authors report a data collection time of several hours for each of the tested vehicle, which would not qualify AutoCAN as a fully automated reverse engineering approach. Nonetheless, the authors have not studied how the performance varies according to the length of the collected trace. We argue that there is ample room for reducing the length of the logs required to few minutes, while maintaining optimal or nearly optimal performance.

Translating signals based on taxonomy is the direct evolution of manual reverse engineering, which makes it sound and reliable. The downside of this approach is that it requires a deep understanding of the CAN data and still a non-negligible amount of human effort and time.

3.4.4 Translation based on companion apps

The goal of this reverse engineering approach is to exploit companion apps, which are mobile or desktop applications that actively interact with in-vehicle networks, to automatically detect and translate the signals.

Wen *et al.* [25] developed CANHunter, the first tool addressing the automation of CAN reverse engineering through the usage of car companion apps. The authors categorise these apps into IVI, and OBD-II dongle apps. The former are the official apps released by car makers and are compatible only with a limited number of vehicles – those of the maker. They connect the mobile phone to the vehicle through cellular network or Bluetooth and offer a variety of features, from starting the engine to controlling the door locks. OBD-II dongle apps are typically developed by aftermarket companies (e.g. insurance companies) and require a dongle attached to the OBD-II port to allow communication between the phone and the CAN bus.

Since each app is built following a different logic and architecture, the first challenge that the authors tackle is to locate the *generation path*, i.e., to find the module sending the requests to the CAN bus and understand its functioning. They solve this problem by developing a software that identifies the standardised network APIs (always present to enable the communication between the phone and the vehicle) and using a technique called *program slicing* to find the source of the CAN requests. After the generation path is understood, it is the turn to address *automatic syntax* and *semantics recovery*, i.e., automatically infer the syntax and semantic meaning of messages. CANHunter achieves syntax recovery through *forced execution*, i.e., the brute-force execution of commands in the generation path with generic inputs. For what concerns the semantics, clues are searched in the User Interface (UI) to relate the outputted strings with the hard-coded commands.

The authors do not test CANHunter on physical vehicles. Instead, they employ 236 free apps – 90 IVI and 146 dongle – that can be downloaded from online app stores, e.g., Google Store. Among those, 107 apps – 3 IVI and 104 dongle – are successfully exploited to identify and decode the semantic meaning of more than 150 000 signals. However, as highlighted by Le Yu *et al.* [26], the majority of these apps are designed for the common driver, who wants to extract ordinary information from the vehicle. As a consequence, the majority of the signals that can be obtained through these apps are responses to the PIDs, for which documentation is already publicly available. In addition, CANHunter does not provide the format of the signals, which does not allow the reuse of the extracted signals.

Le Yu *et al.* [26] propose DP-Reverser, a tool that aims to fix the shortcomings of [25] by exploiting apps that make use of diagnostics communication protocols to obtain the semantic and format of signals that are not publicly known. The authors focus on KWP 2000 (ISO 14230 [80]) and UDS (ISO 14229 [79]), In this work, the authors categorise the apps into (i) *professional handheld diagnostic equipment*, which includes all software and hardware necessary for the communication with CAN, (ii) *professional diagnostic software*, which is a desktop app with a UI that the user can consult, and (iii) *OBD-based telematics app*, a mobile phone app that connects to a diagnostic dongle attached to the OBD-II port.

To deal with the diversity of the diagnostic apps and bypass the security features often present, DP-Reverser makes use of a cyber-physical system emulating human interaction instead of trying to reverse engineer the apps as [25]. This system is composed of two cameras that capture screenshots from the UI and a robotic arm, or *clicker*, which is trained to provide inputs to the platform. The events generated by the clicker are timely correlated with the observed changes in the traffic. For the translation of the format, DP-Reverser employs a genetic programming algorithm. The algorithm starts from a small set of randomly generated formulas to describe the format, and based on the evolutionary principle of the *survival of the fittest*, the next generation of formulas is calculated. The process continues until one of the formula fit the observed behaviour of the signal.

The authors test DP-Reverser on 18 vehicle models and four apps – two professional handheld diagnostic equipment and two professional diagnostic software. A total of 446 signals – 290 physical/counters and 156 status – are identified with an average precision of 98.3 %. Moreover, an accuracy of 100 % is obtained for the format decoding.

Apart from the evident complexity in the initial setting and calibration of all the components of companion apps-based methodologies, the main limitation of this approach concerns the apps themselves, which can be subject to a variety of bugs and errors. In this case, it is unclear how unexpected behaviour of the apps can be handled or if this can potentially drive the algorithms to output wrong decoding results.

Part III

AUTOMATED CAN BUS REVERSE ENGINEERING

4

Tokenisation

In this section, we propose our novel BFR-based tokenisation algorithm which identifies the endianness of the tokens. This algorithm was initially developed as the second phase of CANMatch (see Section 7) and later implemented in FastCAN as well (see Section 5).

4.1 Understanding the endianness

The endianness was firstly introduced by hardware companies. Historically, for instance, IBM used Big Endianness for its architectures, while Intel adopted the Little Endian format. Over the years, the concept of endianness has extended to the point that it can now refer also to the order in which data is sent over a communication channel. In this context, the endianness can also refer to the order of the bits. However, in the case of CAN data, the endianness format sticks to its original meaning: the order of the bytes.

Let us assume a CAN frame payload encoded with the Byte-level Big Endianness format. When reading the payload from left to right, the most significant byte appears first, while the least significant byte appears last. Also the bits within the bytes appear from the most significant to the least significant one. It follows that the overall order of the bits follows the order of the bytes. This format is immediately interpretable by humans because it reflects our daily way of reading decimal numbers. By contrast, according to the Byte-level Little Endian format, the bytes appear from the least significant to the most significant. Differently from

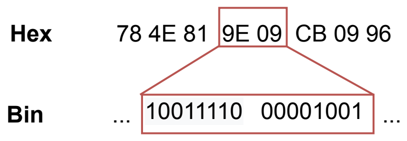


Figure 14: CAN frame from Mercedes A-Class, model year 2018.

Table 4: RPM decoding example with different types of endianness.

	Byte/bit Big Endian	**Byte Little Endian**	**bit Little Endian (reverse bits)**
Parsed Hex	9E 09	09 9E	–
Parsed Bin	10011110 00001001	00001001 10011110	10010000 01111001
Parsed Dec	40457	2462	36985

the BE format, the significance of the bits does not reflect the significance of the bytes. The consequence is that the order of the bits does not follow the order of the Bytes.

This concept is illustrated in Figure 14, which refers to the real payload of a CAN frame extracted from a Mercedes A-Class. The red box contains the Engine Speed signal, whose value indicates the engine RPM at a certain time t. This signal is represented in the LE format. The scale factor is 1 and the offset is 0. Ergo, the RPM is immediately readable once the values are parsed to the decimal format.

Table 4 compares different interpretations of the endianness for the signal. Firstly, we consider the Byte-level Big Endianness, which also corresponds to the bit-level Big Endianness, as explained above. By using this format, the RPM value obtained is clearly wrong, as the engine of a commercial car cannot possibly reach 40457 RPM. Then, we consider the Byte-level Little Endianness. According to this format, the first byte is the least significant one and the last byte is the most significant one. Therefore, if we want to read the signal from left to right, we need to invert the order of the bytes and, only then, parse to decimal. In this case, the value of the RPM is correct (2462) and also matches our expectations regarding the RPM.

Finally, we assume bit-level Little Endianness. This is what Nolan et al. erroneously do [107]). In this case, the bits of the whole signal are completely reversed. Following this approach, in our example, when reading the signal from left to right, we encounter: i) the least significant bit of the most significant Byte (pos 0), ii) the most significant bit of the most significant Byte (pos 7), iii) the least significant bit of the least significant Byte (pos 8), iv)

Algorithm 1 Tokenisation

```
Input: Input Trace $I$
Output: A set of tokens $T$
 1: $T \leftarrow []$
 2: for $id$ in get_frame_IDs($I$) do
 3:     $ts \leftarrow$ get_frame_time_series($I$, $id$)
 4:     $bfr \leftarrow$ calculate_bit_flip_rate($ts$, $I$)
 5:     $be \leftarrow$ tokenise_big_endian($bfr$)
 6:     $le \leftarrow$ tokenise_little_endian($bfr$)
 7:     $be\_types$, $le\_types \leftarrow$ get_tokens_type($be$, $le$)
 8:     if score($be$) $\geq$ score($le$) then
 9:         $T$.append($\langle be, be\_types, 0\rangle$)
10:     else
11:         $T$.append($\langle le, le\_types, 1\rangle$)
12:     end if
13: end for
```

the most significant bit of the most significant Byte (pos 15). As expected, the parsing to decimal representation results in a wrong interpretation of the signal value.

4.2 Methodology

The tokenisation process is based on the same assumption made by Marchetti and Stabili [28]: differences among two consecutive values of a signal representing a physical phenomenon are small, due to the own nature of the phenomenon. Following this principle, the least significant bit, which impacts the least on the value of the signal, usually flips (changes its value) more often than a more significant bit of the same signal.

The pseudocode summarising the implementation of our proposal is illustrated in Algorithm 1. The algorithm extracts all the frame IDs from the trace and splits the trace into n sub-traces, where n is the number of unique IDs in the trace (line 3). Each of these sub-traces contains the payload of frames associated to the same ID, in the same temporal order as they appear in the trace. Then, the sub-traces are iteratively processed independently of one another. For each sub-trace, the BFR of each bit in the frame is extracted (line 4). Once all BFRs within a frame are computed and stored in an array according to their position in the frame, the algorithm proceeds extracting the boundaries and endianness of the tokens.

Regarding the endianness, following [107], these assumptions are made:

- A CAN trace can contain signals encoded with both big and little endian formats.
- Signals within a single frame are encoded using the same format.

A set of tokens is extracted assuming one endianess format at a time (lines 5–6). For the BE encoding, the BFR array is scanned from the beginning to the end, seeking for a drop in the BFR between consecutive elements. A drop in the BFR value is considered if the difference between two consecutive elements is higher than a tolerance threshold τ:

$$|a_i - a_{i+1}| > \tau \tag{2}$$

where a_i is the i^{th} element of the array. When such a decrease is found, a boundary between two tokens is identified. The goal of τ is to prevent the premature identification of boundaries between two tokens caused by small statistical fluctuations in the BFR.

For the LE format, the algorithm first reverses the bits in each individual byte, without changing the order of the bytes. This approach is not a try to reconstruct a priori the correct order of the bytes. Instead, its aim is to exploit the technique presented in [23], [28] of identifying continuity (or disruption) in the BFR along the whole frame also in the case of a LE format. In fact, as for BE, the array is scanned to find decreases in the BFR values. But, due to the different order of the bytes of the LE format, the scan is performed from the end to the beginning of the BFR array. For the sake of comprehension, it should be noted that the reversing of bits within the bytes is exclusively carried out with the purpose of evaluating the likelihood of the two endianness formats. As a matter-of-fact, after the tokenisation is performed, the BFR array is brought back to its original form and the boundaries are updated accordingly. Figure 15 shows an example of tokenisation on a 2-Byte payload performed according to the two endianness encodings.

Once the tokenisation is completed, the algorithm pursues to identify the token types in the two sets (line 7). Multi-value, checkcode and constant tokens are labelled in a similar fashion to [23], [28] presented in Section 3.3. By employing simple techniques for time series

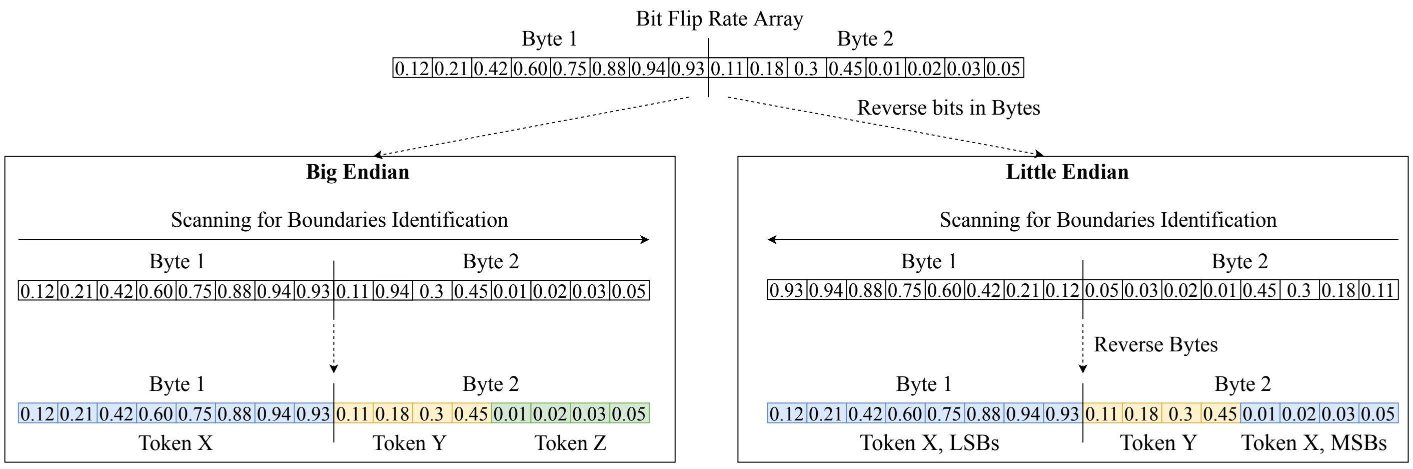


Figure 15: Tokenisation according to different types of endianness: (i) assuming big endian (left figure), two distinct one-byte-long tokens are found; (ii) assuming little endian (right figure), one two-byte-long token is found.

analysis, the algorithm is able to identify counter tokens according to the following properties: (i) counters that exhibit a seasonal trend, and (ii) counter cycles that are characterised by a monotonic growth followed by a sudden drop (the counter is reset). Tokens that do not fall in any of the previous categories are labelled as physical.

Once the token types are found, the algorithm decides which endianness is the correct one and, subsequently, which is the final set of tokens to extract from the frame. For this purpose, a *likelihood score*, S_{L}, is calculated on the tokens of both outputs. S_{L} depends on two main components (subscores): (i) the number of cross-byte tokens, and (ii) a volatility component that penalises highly-volatile tokens.

To find the first subscore, the algorithm computes a preliminary score S_{c}, defined as:

$$S_{\mathrm{c}} = |T'| + \alpha \sum_{t \in T'} (S_t + E_t) \tag{3}$$

Here, T' corresponds to the set of cross-byte tokens (i.e., tokens contained in different consecutive bytes). We expect the tokenisation algorithm to find a higher number of these signals when the endianness is the correct one. S_t and E_t are binary variables defined as follows:

$$S_t = \begin{cases} 1, & \text{if } start\_bit_t \pmod 8 = 0 \\ 0, & \text{otherwise} \end{cases} \tag{4}$$

$$E_t = \begin{cases} 1, & \text{if } (end\_bit_t + 1) \pmod 8 = 0 \\ 0, & \text{otherwise} \end{cases} \tag{5}$$

To be noted that the second component of Equation (3) is weighted by α in order to incentivise tokens that start and/or finish exactly at the least and/or most significant bit inside the frame. While the CAN protocol allows signals to start and end at any position within a byte, the start and end bits of long signals often coincide with, respectively, the least and most significant bit of a byte. S_c is then normalised between 0 and 1. We call this normalised score S'_c.

The volatility component, S_v, can be computed as follows:

$$S_\text{v} = \frac{1}{|T'|} \sum_{t \in T'} V_t, \tag{6}$$

where V_t corresponds to the volatility of a single token in T'. It is in the range [0,1], where values that tend to 1 indicate a static behaviour, while values that tend to 0 express a highly volatile behaviour. The reason we consider the volatility component is because signals encapsulate information that has a physical meaning. If a token is correctly extracted, i.e., its boundaries and endianness are correct, its time series perfectly describes the behaviour of the vehicle function/telemetry it carries over time. By contrast, cross-byte tokens that are not correctly extracted usually contain bits belonging to different signals. The time series of such tokens do not represent any physical information and, as a consequence, are characterised by a highly volatile and unpredictable behaviour. Our assumption is that, among different sets of tokens computed for the same frame ID, the set that produces tokens whose time series are less volatile on average has a higher chance to be the correct one.

Finally, the likelihood score is computed as follows:

$$S_{\mathrm{L}} = S'_{\mathrm{c}} + \beta \times S_{\mathrm{v}}, \tag{7}$$

where β assigns a weight to the volatility component.

4.3 Performance Evaluation

In the following, we present the dataset employed for the evaluation and the results achieved by our tokenisation algorithm.

4.3.1 Data Collection

As raw CAN traces to be tokenised, we use two sets, as follows:

- Driven Vehicles Traces (DVT) – a set of 13 CAN raw traces. These traces are 1-2 min long and are collected on moving vehicles. The ground truth of each of these vehicles is contained in SDDBC.
- OpenDBC DVT (ODVT), a set of 2 CAN raw traces. These traces are 1 min long and are collected on moving vehicles. The ground truth of each of these vehicles is contained in OpenDBC.

Information on DVT and ODVT is presented in Table 5.

All traces in DVT were extracted with a `PCAN-USB FD` dongle [48], while traces in ODVT were collected with different dongles. The majority of traces were collected through the OBD-II. However, when a gateway was present, we identified and attached the dongle directly to the CAN bus wires with the use of a clip. Finally, all tests presented in the following sections are performed under real-world conditions, but the results refer only to the decoded signals present in the ground truth.

Table 5: DVT and ODVT set

Set	Vehicle Model	# Signals DBC	Length (s)
DVT	Audi A3 2012-	59	60
	BMW X1 2015-	53	60
	Citroen C3 Picasso 2009-	46	120
	Fiat Qubo 2008-2019	26	120
	Kia Sportage 2016-	31	60
	Mercedes A-Class 2018-	59	60
	Nissan Micra 2005-2011	41	120
	Peugeot 208 2011-	47	120
	Peugeot 307 2005-2008	37	60
	Renault Captur 2013-	46	120
	Renault Megane 4 2016-	48	60
	Smart Fortwo 2014-	57	120
	VW Golf 5 2003-2009	63	60
ODVT	Acura ILX 2013-	177	60
	Volvo V40 2017-	54	60

4.3.2 Results Analysis

For what concern, the evaluation, we adopt the metrics defined by Pesé *et al.* [23], TDBC and TE. However, in their work, TDBC corresponds to the actual total number of signals that can be found in each of the four tested vehicles used for its evaluation, as the DBC files used for as ground truth were obtained from the actual OEM. In our case, we do not have a complete ground truth for each of our testing traces and, due to this limitation, we evaluate the tokeniser using the ratio CE/TDBC.

No status/multi-value signals were intentionally triggered by the driver at data collection time and, therefore, their bit flip cannot be possibly observed. Since such signals are typically 1–4 bit long, in the evaluation we consider only signals whose length is equal or greater than 5 bit.

Let S be the set of known signals in the ground truth. In general, a token t is considered to be correctly identified if exists a signal $s \in S$ such that the boundaries of t and s coincide, i.e., their start/end bits are the same. In our case, we allow a 4 bit tolerance error for the most significant bit in response to a widely known problem that affects tokenisation [23], [28], [30]. For many signals – especially those representing physical information – it is very difficult to record flips in their most significant bits, as they can be activated only under exceptional (and often extreme) circumstances. For instance, the maximum theoretical speed of a car is hardly recorded, unless the car is driven on purpose to reach that speed. While no error can be tolerated in the identification of the start bit of a token, as highlighted by Verma *et al.* [24], since it would inevitably affect the decoding of the scale factor and offset, we argue that a certain tolerance error is acceptable for the end bit. As a matter of fact, cutting off the most significant bits of a signal might not allow us to correctly interpret the maximum value that can be reached by the signal, as intended by the manufacturer. Nonetheless, it does not affect the interpretation of the signal when it carries physical values under "normal" driving conditions. We chose 4 bit as tolerance error, because it seems a good trade off between addressing the described issue and a fair evaluation of our algorithm.

We first performed an extensive tuning on 8800 combinations of the parameters α, β, and τ, in order to find the one that guarantees the best CE/TDBC ratio. Figure 16 presents the average results obtained on all tested vehicles with every combination of α and β, and $\tau = 0.3$. To be noted that we evaluated other values of τ as well, but present here only the best value. The results show that a maximum mean CE/TDBC score of 77.6 % is achieved with $\tau = 0.3$, $\alpha = 0.2$, and $\beta = 0.2$. This suggests that the number of long tokens constitutes the most valuable contribution for the likelihood score.

Figure 17 compares our proposal with READ [28] and the tokenisation algorithm of LibreCAN [23], which, at the time of writing, are the start-of-the-art algorithms for tokenisation. The results show that our proposal equals or outperforms READ and LibreCAN on all considered vehicles of DVT. The difference in the performance, especially remarkable for some of the vehicles, is driven by the presence of LE cross-byte signals – signals whose information is encapsulated in multiple (consecutive) bytes. The cross-byte signals are typically signals which carry physical information and, therefore, require a large number of bits to be represented, e.g. signals related to throttle pedal, steering wheel, engine etc. Failing to take into account the endianness during tokenisation can result into splitting a cross-byte signal into two (or more) signals, as in the example illustrated in Figure 15.

Our analysis on SDDBC shows that, out of 4667 cross-byte frames, 65.3 % of the signals are represented according to the BE format and 34.7 % are represented according to the LE format.

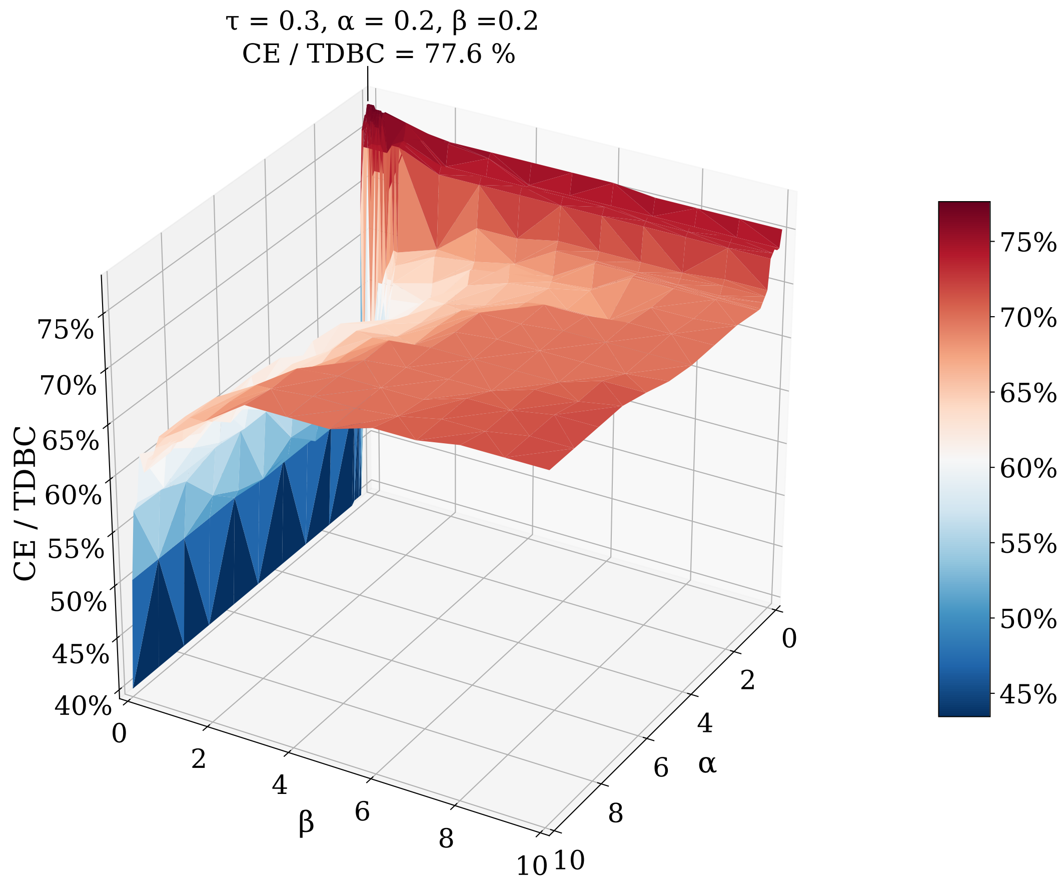


Figure 16: Tuning of weight parameters α, β, and τ.

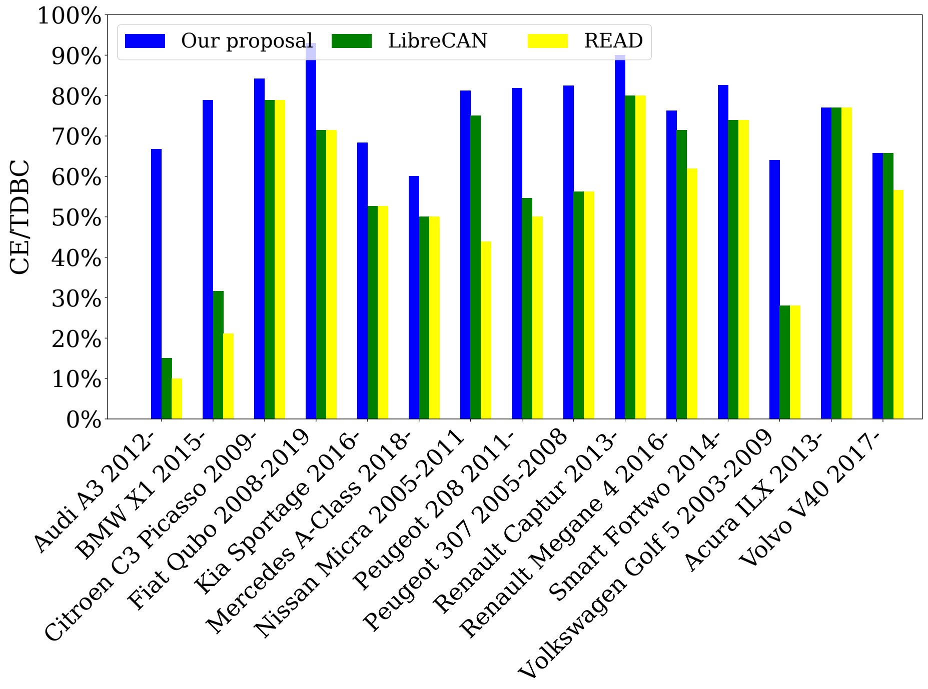


Figure 17: Phase 2 performance evaluation: comparison with the state of the art.

Table 6: Endianness impact on tokenisation.

	Recall	**Precision**
Big Endian	85.2 %	99.2 %
Little Endian	60 %	100 %

We also observed that all signals within a frame are represented with the same endianness. Our tokenisation algorithm exploits this information to correctly identify the endianness of all cross-byte signals and, subsequently, their correct boundaries.

Table 6 shows how endianness impacts the performance of the tokenisation process on DVT. On the one hand, Phase 2 is able to identify tokens represented in BE format better than tokens represented in LE format. On the other hand, all tokens labelled as LE are actually in LE format, while a small rate (0.8 %) of tokens labelled as BE are, instead, in LE format.

Phase 2 achieves a performance similar to LibreCAN and READ on ODVT. The reason is the absence of LE signals in this set, as reported in the generated ground truth available in [52]. It is still worth noting that, in this case, the search for LE signals does not impact negatively the capability of Phase 2 of correctly identifying BE signals. This further corroborates the findings presented in Table 6.

5 A Methodology for Semi-Automated Reverse Engineering

Semi-automated CAN reverse engineering has been shown to provide decoding accuracy comparable to the manual approach, while reducing the time required to decode the signals. However, current approaches are invasive, as they make use of diagnostic messages injected through the On-Board Diagnostics (OBD-II) port and still take up to one hour for the data collection [23], [83].

In this section, we present a pipeline to perform semi-automated CAN bus reverse engineering methodically and at an unprecedented speed. The novelties introduced in this work concern the data collection and the translation, which are driven by an extended categorisation of the CAN signals. We validate this approach by testing FastCAN, a novel tool based on the proposed pipeline.

5.1 Taxonomy of the Signals

CAN signals are typically divided in the five main categories identified by [23], [24], [28] (see Section 2.2): physical, status, counters, checksums and unused/constant. The signals represented by each of these categories display behaviours that are intrinsically different from one another. A reverse engineering approach looking for correlations between the signals and

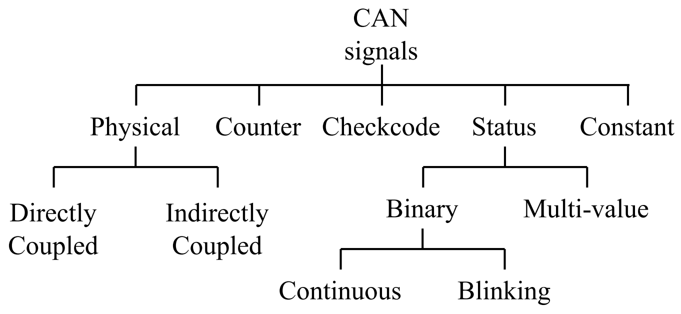


Figure 18: Extended taxonomy of CAN signals.

the real world should be adapted to take into consideration the peculiarities of the signals encapsulating the target vehicle functions.

To optimise the reverse engineering process it is fundamental to preliminarily understand the nature of the signals and to identify the category to which they belong. For this reason, the starting point of our methodology is the taxonomy of the signals. In this scope, we propose an extension of the taxonomy presented in Section 2.3.

We suggest the division of physical signals in two subcategories: *directly coupled* and *indirectly coupled*. When considering a dynamic driving scenario, all physical signals in one way or another influence each other. However, the degree of mutual impact between any two signals depends on the real world relation between the vehicle functions that they encapsulate. Directly coupled signals exhibit a similar behavior among them (e.g. wheel speed) and/or are tied by a clear principle of cause and effect (e.g. the throttle pedal position and the engine RPM). By contrast, indirectly coupled signals are those whose correlation with any other physical signal has to be inferred in a non-trivial way (e.g. brake pedal position and engine RPM).

According to the literature, status signals can be subdivided in two sets, binary (on/off) and multi-value. We propose to further sub-categorise binary signals into *continuous* and *blinking*. Once triggered, the former maintain their new value (until the status changes again), while the latter periodically change their value from on to off and vice versa.

Figure 18 shows the novel taxonomy of the signals proposed in this work.

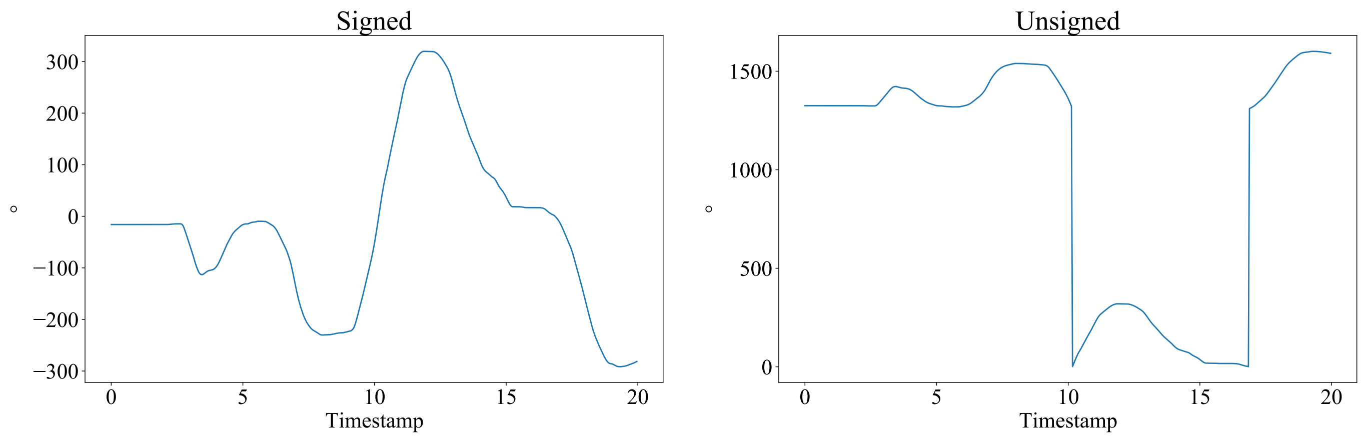


Figure 19: Comparison between the correct interpretation of the signedness of a signal related to the steering wheel angle (left) with the same signal interpreted as unsigned (right).

5.1.1 Translation of Physical Signals

The suggested approach is to initially collect a CAN trace for each group of directly coupled vehicle functions and, then, proceed with an individual log for each of the vehicle functions left out. Specific actions have to be performed to capture the dynamic behavior of the searched signal or group of signals and possibly reduce the noise produced by other signals. For instance, to trigger and isolate signals related to the speed of the four wheels it is convenient to drive the vehicle in clockwise or anticlockwise circles. After applying tokenisation on such a trace, a GPS trace can be employed to identify all the signals related to the vehicle speed. Then, the individual wheel speeds can be translated by finding differences in the speed values, e.g. by driving clockwise the front left wheel is the fastest while the rear right wheel is the slowest.

Physical signals can be unsigned or signed, based on whether they can assume only positive or negative values (see Section 2.2). An efficient way to assess the signedness of a signal is to parse its information into raw unsigned decimal values throughout the trace, and spot eventual incoherent behavior. Figure 19 shows an example of the impact that the signedness has on the interpretation of the a signal values. A signal time series characterised by sudden changes in the values is likely caused by the misinterpretation of the most significant bit.

For the interpretation of the format, a suitable approach is to feed a linear regressor with the decimal raw data of the signal and external ground truth (e.g. GPS data) or another signal whose format has been already decoded (see Section 7.1.3).

5.1.2 Translation of Status Signals

Similarly to physical signals, one trace has to be collected for an individual status signal or group of them. Also, a trace where no operation is performed, called reference trace, must be logged to provide the ground truth to compare any other trace with and, thus, identify changes in the CAN traffic as proposed by Pesé *et al.* [23]. Usually, while it is easy to spot these changes, correctly identifying the signals of interest is not straightforward. In fact, a single action in the vehicle can trigger multiple signals other than the ones currently researched.

Noise can be preliminarily reduced by discarding signals based on temporal patterns that cannot reflect a human action, e.g. a signal switching on and off in 0.1 s cannot possibly represent a door being opened and closed.

When a subset of candidate status signals is identified for a set of vehicle functions, it is necessary to understand whether they are continuous or blinking. For this task, it is essential to preliminarily identify their default value (the value of the signal when it is not triggered), which can be 0 (inactive) or 1 (active).

5.1.3 Translation of Counters

Counters are characterised by a monotonic growth followed by a sudden drop in their value (reset). It is to be noted that the sudden drops in the value are a feature in common with physical signals whose signedness has not been decoded yet (see Figure 20). The risk is to interpret a counter as a signed physical signal. When the presence of cyclicity cannot be assessed, i.e. the considered log is too short, other heuristics should be considered to prevent this risk, e.g. a counter does not decrease in value unless it is the drop related to its reset.

5.1.4 Translation of Checksums and Identification of Constant Signals

When physical, counter and status signals have been translated, the checksums can be identified with any of the state-of-the-art BFR-based methodologies presented in the literature [23], [28],

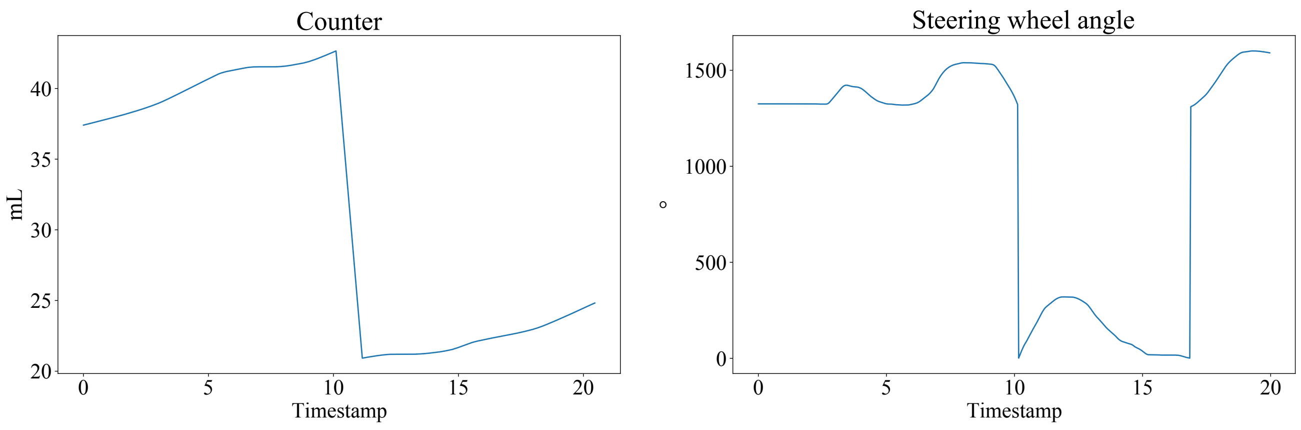


Figure 20: Comparison between the time series of a fuel consumption counter and a signed steering wheel angle signal whose signedness has not been assessed.

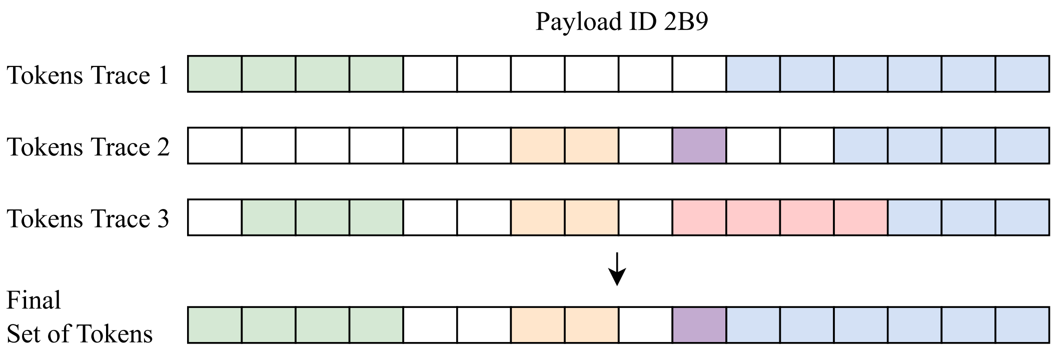


Figure 21: Example of set of tokens extracted from frames wit ID 2B9, based on the output of tokenisation runs on different traces. It is to be noted that longer signals are chosen over multiple signals occupying the same position, as in the case of the signal represented in blue.

[29].

Finally, constant/unused signals can be found by aggregating the results of the tokenisation runs on all the collected traces. Executing the tokenisation on traces from multiple driving sessions likely results in finding slightly different sets of tokens, due to the fact that the most significant bits of some signals might be stimulated more in a scenario rather than in others. By joining all the outputted set of tokens it is possible to narrow down on the bits that never flip. Figure 21 shows an example of identification of constant signals in a payload, starting from the tokens extracted from three traces used to decode different set of signals.

5.2 Performance Evaluation

We have designed FastCAN, a CAN bus reverse engineering tool able to decode signals carrying a total of 24 distinct vehicle functions, based on the presented methodology. It is to be noted that a number of status signals are not present in some vehicles. For instance, high-end cars typically have one dedicated ECU for each seat belt, while some low-end cars have only one for the front-left seat belt. For this reason, we assess the performance based on the signals that can actually be found in each vehicle.

To test FastCAN, we have collected 10 CAN traces for each of 5 vehicle models from 5 distinct manufacturers. Three of these traces are collected to decode the following physical signals: vehicle/wheels speed (T1), throttle pedal position/engine RPM (T2), and steering wheel angle (T3). One trace is to decode the fuel consumption counter (T4). Six traces are used for status signals: doors (T5), seat belts (T6), indicators (T7), air conditioning (T8), and wipers (T9). One trace is the reference trace. The data collection was conducted using a Raspberry Pi 3, equipped with a Pican 2 Duo [50] interface and synchronised with a Xee Connect [51] providing GPS speed information. The hardware equipment is shown in Figure 22.

For the validation, we employ the generated DBC files presented in Section 2.5. Since we cannot precisely assess whether tokens absent in the ground truth would be correctly classified, we solely present the results of our evaluation based on the signals present in our DBC files. For this reason, we consider the recall and the False Positive Rate (FPR) of known signals. The recall corresponds to the number of correctly translated signals over the number of signals to translate for each vehicle function, as described in the ground truth. For each vehicle function, the FPR is the number of other mislabelled signals over the total number of other signals. The Normalized Root Mean Squared Error (NRMSE) is the difference between the time series of the physical signal parsed with its reference format (i.e. ground truth factor and offset) and the calculated format (i.e. outputted factor and offset), on a test CAN trace. Tables 7 and 8 present the aggregated performance obtained on all vehicles.

With an average of 25 s per trace, or 5 min in total, the data required to identify and

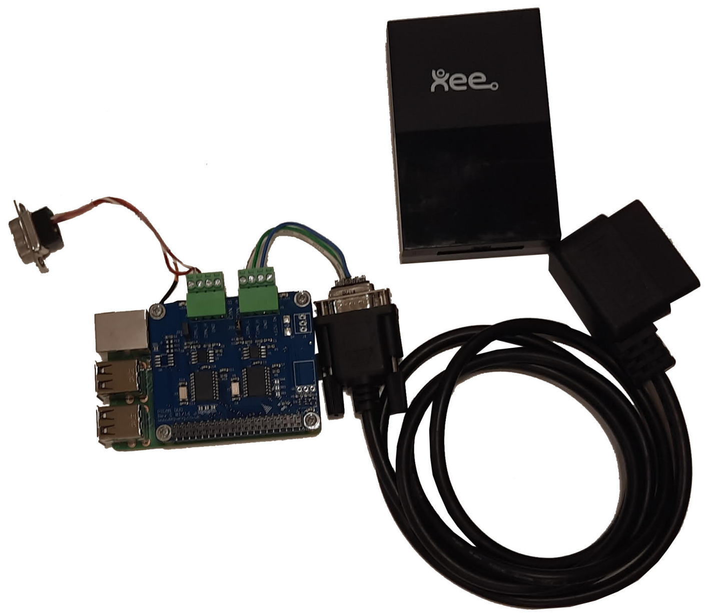


Figure 22: Hardware employed by FastCAN for the data collection: a Pican 2 Duo for the logging of CAN data and a Xee Connect for GPS/IMU data.

Table 7: Evaluation physical and counter signals

	T1	T2	T3	T4
Mean Recall	85.0 %	88.8 %	66.7 %	80.0 %
Mean FPR	1.8 %	0.7 %	0.1 %	0.9 %
Mean NRMSE	2.1 %	4.9 %	4.5 %	1.0 %

Table 8: Evaluation status signals

	T5	T6	T7	T8	T9
Mean Recall	95.8 %	72.8 %	57.1 %	80.0 %	87.5 %
Mean FPR	0 %	0 %	0 %	0 %	2.5 %

decode up to 24 vehicle functions is one order of magnitude inferior with respect to current state-of-the-art solutions, while achieving similar performance [23], [24].

5.3 Discussion

We presented a methodology to perform fast semi-automated CAN bus reverse engineering, based on the taxonomy of CAN signals. Particular attention is paid to the data collection process, which is decomposed in multiple steps and reflects systematically the semantic of the researched signals. In addition, we offered indications on how to identify and translate signals based on their characteristics and format. We validated this approach by implementing a CAN bus reverse engineering tool based on it, FastCAN. FastCAN requires significantly less time for the data collection compared to other state-of-the-art solutions. Future work includes testing on a wider number of signals and further optimisation of the reverse engineering process.

6

Critical Signals Identifier

A number of methodologies for fully automated CAN reverse engineering have been presented in the literature. While achieving convincing results, these tools are invasive, i.e. recur to the injection of diagnostic messages, make use of external IMU/GPS data and/or require complex hardware configurations inside the vehicle. In this section, we investigate the possibility of achieving fully automated translation of the signal semantics with minimal hardware requirements, i.e. only a dongle for the passive logging of the CAN data.

Jaynes *et al.* [54] are the first to aim at translating the semantic meaning of the CAN data. However, they train ML classifiers to translate the semantics of the entire frames. This approach has two non-negligible limitations, (i) a single label for the whole frame does not capture the nature of the signals composing its payload, (ii) it is not scalable, i.e. semantically similar frames in any two vehicle models do not necessarily have the same signals composition.

Hereafter, we describe CSI, a first attempt to translate the semantic meaning of CAN signals through ML classification. In this study, we work under limited assumptions. Instead of trying to reverse engineering any kind of signal, we focus on translating 10 signals, which we consider of primary relevance in a vehicle and critical for its safe operation. These functions are: engine speed, vehicle speed, wheel speeds, steering wheel angle, steering wheel side, throttle pedal position, battery voltage, engine coolant temperature, engine oil temperature, and odometer.

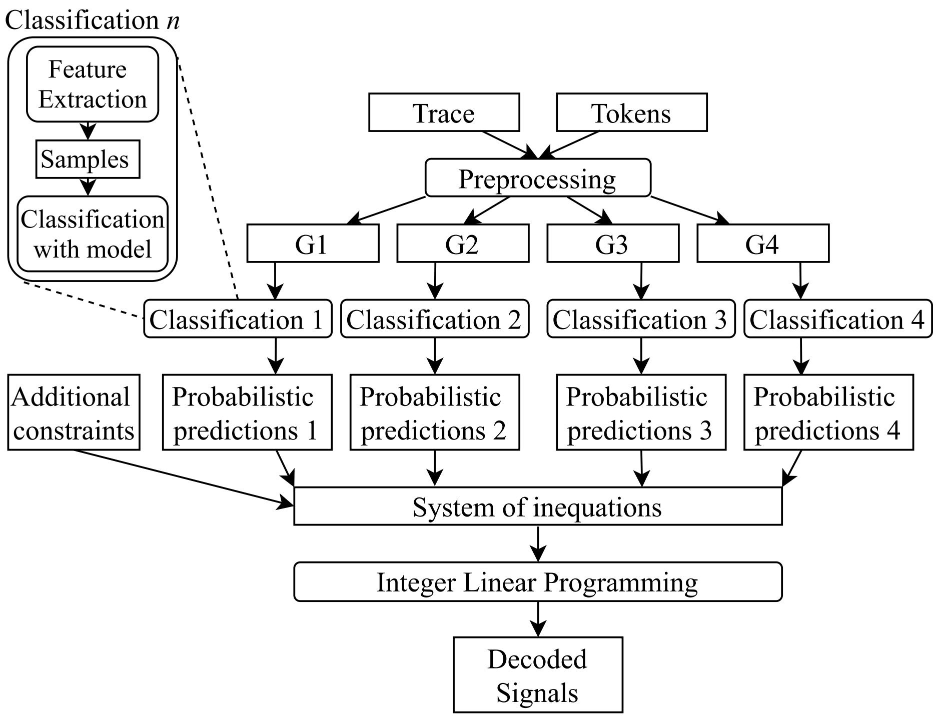


Figure 23: Pipeline of CSI.

The goal of CSI is to identify these critical signals, referred to as *foreground set*, among a wider set of signals, the *background set*, from a trace of raw CAN data. The critical signals are not obtained through tokenisation as we assume the raw data is already divided into tokens before running the CSI algorithm.

6.1 Pipeline

Figure 23 illustrates the entire pipeline of our proposed solution. The CSI classification task is performed by ML models, trained on known data related to previously reverse engineered vehicles, which are applied to a trace of raw CAN data extracted from the vehicle we want to reverse engineer.

6.1.1 Signal Categories

We assume signals representing the same vehicle functions have identical or similar length across vehicle models, due to the intrinsic amount of information they need to encapsulate. To validate this assumption, we conducted an analysis on a set of the SDDBC dataset (see Section 2.5). We discovered that about half of the signals related to the same telemetry have exactly the same length in all the vehicles. In addition, all considered vehicle functions have a Coefficient of Variation (CV) related to their length less than 0.5. CV is a normalised measure of the dispersion of frequency distribution, computed as standard deviation to mean ratio ($\frac{\sigma}{\mu}$). CV < 1 indicates a low variance, while CV ≥ 1 indicates a high variance.

As a result, we decided to divide the signals into four groups, according to their length: *G1* (1–4 bits), *G2* (5–10 bits), *G3* (11–17 bits), and *G4* ($\geq$ 18 bits). We tackle the reverse engineering process as four different classification tasks. The goal is to design features that better fit each group and build a more accurate model for the classification task.

6.1.2 Feature Extraction

The aim of a supervised ML classifier in the scope of CAN bus signal translation is to identify for each token the vehicle function that it represents. The classifier is trained to associate a sample representing each token with its vehicle function. Each sample is a combination of feature values calculated for a token. A feature describes a unique characteristic of a vehicle function. The features should make signals related to the same function recognisable across different vehicle models while differentiating them from other signals.

The main difficulty in finding good features for CAN signals is that most of them are directly or indirectly highly influenced by environmental factors, such as road/traffic conditions and the driving style. The risk is of identifying features that are not related to the intrinsic nature of the signal but, instead, are strongly influenced by the environmental factors. In related work, sensors such as GPS and IMU provide some useful ground truth about the driving context. For instance, by cross-correlating the speed pattern recorded by a GPS dongle with a set of tokens, it is possible to identify the signals related to the vehicle speed. In our use-case scenario, we do not have access to such ground data to match the CAN raw trace with. Therefore, we cannot make the signals comparable by scaling their values, given the absence of a reference truth.

To identify the set of features, we initially tried to exploit the correlation among signals across three domains: frame priority (inferred from the CAN ID), frame sending frequency, and payload. However, our preliminary analysis of the dataset revealed that the first two domains do not present high enough correlation values. For this reason, we focus only on the payload carried by each signal. The numerical values carried by the payload of each signal strongly depend on contextual factors and do not constitute a good base for fingerprinting the vehicle functions. For example, a model trained on traces related to cars going at most at 30 km/h is unlikely to recognise the vehicle speed in a trace from a car reaching 100 km/h.

To minimise the impact of these factors, we design features that are categorical. These features describe either (i) the presence or absence of a property (e.g., twinned or not), or (ii) a categorisation or ranking of the signals inside the same trace with regard to a certain aspect (i.e., dynamicity rankings). These features fit the vehicle functions independent of the numerical values carried by the signals representing them.

We define a total of eight features, which require several processing steps to be generated from the CAN raw trace:

- **Number of Static Bits:** Unless the vehicle is pushed to its extreme performance with regard to a telemetry, some bits of the signal carrying this telemetry will never flip. The number of unchanging bits helps to identify signals carrying these telemetries.
- **Length:** The length of the signal itself is used as a feature.
- **Signal Type:** In this work, we consider only physical, status and counter signals.
- **Short Signal Dynamicity Ranking:** It is related to status signals. The feature ranks the dynamicity of the signals from "Low" to "High" based on the bit flip rate percentiles calculated taking into consideration signals of the same type found in the trace.
- **Long Signal Dynamicity Ranking:** Same as Short Signal Dynamicity Ranking, but

Table 9: Features for each group of samples.

Feature	**G1**	**G2**	**G3**	**G4**
Number of Static Bits		✓	✓	
Length	✓			✓
Short Signal Dynamicity Ranking	✓			
Long Signal Dynamicity Ranking		✓	✓	✓
Dynamicity Type	✓	✓	✓	✓
Stopping Dynamicity		✓	✓	
Idle Status		✓	✓	
Twinned			✓	

related to physical signals and counters.

- **Twinned:** Two or more signals within the same frame are twinned if they display a similar bit flip rate. This is used to identify wheel speed signals.

- **Stopping Dynamicity:** We assume the vehicle to reverse engineer is driven for a certain amount of time until it stops. We identify the parts of the trace related to the two phases of the driving session (driving and not driving) and find the speed-related signals whose values decrease down to the idle value. This feature encapsulates the temporal order in which these signals reach their idle status.

- **Idle Status:** This feature captures the idle status of the signals (i.e., their value), when the vehicle is not moving. This feature is useful to further discriminate speed related signals from the others, such as the throttle pedal position.

The samples of each group of signals are composed of a subset of these features, as shown in Table 9.

6.1.3 Classification

Once the samples are generated for each token according to the features of each group, they are passed to an ML model for the classification task, as shown in Figure 23. There are four ML models, each trained on samples of one of the four groups extracted from traces of previously reverse engineered vehicles.

Some signals sent within the CAN bus encapsulate the same vehicle function. This is especially true for critical signals such as vehicle speed and throttle pedal position. These signals, carrying the same vehicle function, hereafter referred to as *redundant*, are typically represented with a different format and/or refer to another measurement unit (e.g. mi/h and km/h for the speed). In this work, we assume that the number of redundant signals for each vehicle function. We define this number as the *cardinality* of the vehicle function.

To the best of our knowledge, no classifier can be set to take into account the constraints during the prediction. Since the number of samples per class is not known a priori, in case of misclassification, a standard classifier would attribute the same label to an arbitrary number of tokens.

To overcome this limitation, instead of performing deterministic predictions, we set the classifier to output a list of probable labels/functions for each token. Subsequently, CSI converts each function predicted for each token into a variable, and the probability associated with it into a coefficient associated with the variable. All the variables are inserted into a system of inequalities, describing the constraints related to the cardinality of the vehicle functions. Then, the system is solved as a ILP problem, described as follows:

$$\text{maximise} \sum_{t \in T} \sum_{f \in F} \alpha_{t,f} p_{t,f} \tag{8}$$

$$\text{subject to:} \sum_{t \in T} p_{t,f} \leq c_f, \forall f \in F \tag{9}$$

$$\sum_{f \in F} p_{t,f} \leq 1, \forall t \in T \tag{10}$$

$$0 \leq \alpha_{t,f} \leq 1 \tag{11}$$

$$c_f \geq 1 \text{ integer} \tag{12}$$

$$p_{t,f} \in \{0,1\} \tag{13}$$

where T represents the set of tokens, F is the set of vehicle functions, $\alpha_{t,f}$ indicates the probability that token t contains the vehicle function f as predicted by the classifier, c_f

represents the cardinality of the vehicle function f, and $p_{t,f}$ is a binary function defined as:

$$p_{t,f} = \begin{cases} 1, & \text{if the vehicle function } f \text{ is ultimately assigned to token } t \\ 0, & \text{otherwise} \end{cases} \tag{14}$$

The constraints described by Equation (9) ensure that at most c_f tokens get associated with a certain vehicle function f, while Equation (10) ensures that each token is ultimately predicted to contain with at most one known vehicle function.

6.2 Performance Evaluation

To evaluate the performance of our proposed solution, we used a set of 60 s-long CAN traces collected with a `PCAN-USB FD` [48] dongle connected to the CAN bus of the following vehicle models: Audi A3 2012, BMW X1 2015, Kia Sportage 2016, Mercedes A-Class 2018, Peugeot 307, 2008, Megane 4 2016, Volkswagen Golf 5 2009 and Volvo XC40 2018. The ground truth for the content of these CAN traces was obtained through manual reverse engineering by Xee. To obtain more samples to train the classifiers better, each trace is divided in 10 smaller traces of about 6 s each. A total of 2,302 signals were found in these 80 subtraces. The background set is composed of signals related to 59 different vehicle functions.

For the classification task, we tested four of the most used ML classifiers in the related work: SVM, Naive Bayes classifier (NB), Random Forest (RF), and Multilayer Perceptron (MLP). For the implementation of the classifiers, we opted for the popular implementations offered by the library Scikit-Learn [111]–[114]. To cross-validate CSI, we trained iteratively each classifier on samples belonging to all but one vehicles and used the samples of the remaining vehicle as a test set. To optimise the performance of each classifier, we balanced the training set with SMOTE [115], a well-known technique for over-sampling the minority classes, and we extensively tuned its hyperparameters. For what concerns the formalisation and the resolution of the ILP problem, we employed PuLP, a popular Python library for Linear Programming [116].

The evaluations presented in the following refer to the combined predictions achieved on

Table 10: CSI performance for different classifiers compared to Baseline.

Performance metric	**CSI**				**Baseline**
	RF	MLP	SVM	NB	RF
Accuracy [%]	91.6	93.0	91.7	89.3	12.7
Balanced Accuracy [%]	73.0	80.0	72.4	63.4	12.8
F1-Score [%]	91.7	92.8	92.1	89.2	11.5
Balanced F1-Score [%]	74.1	79.9	74.4	59.9	11.4

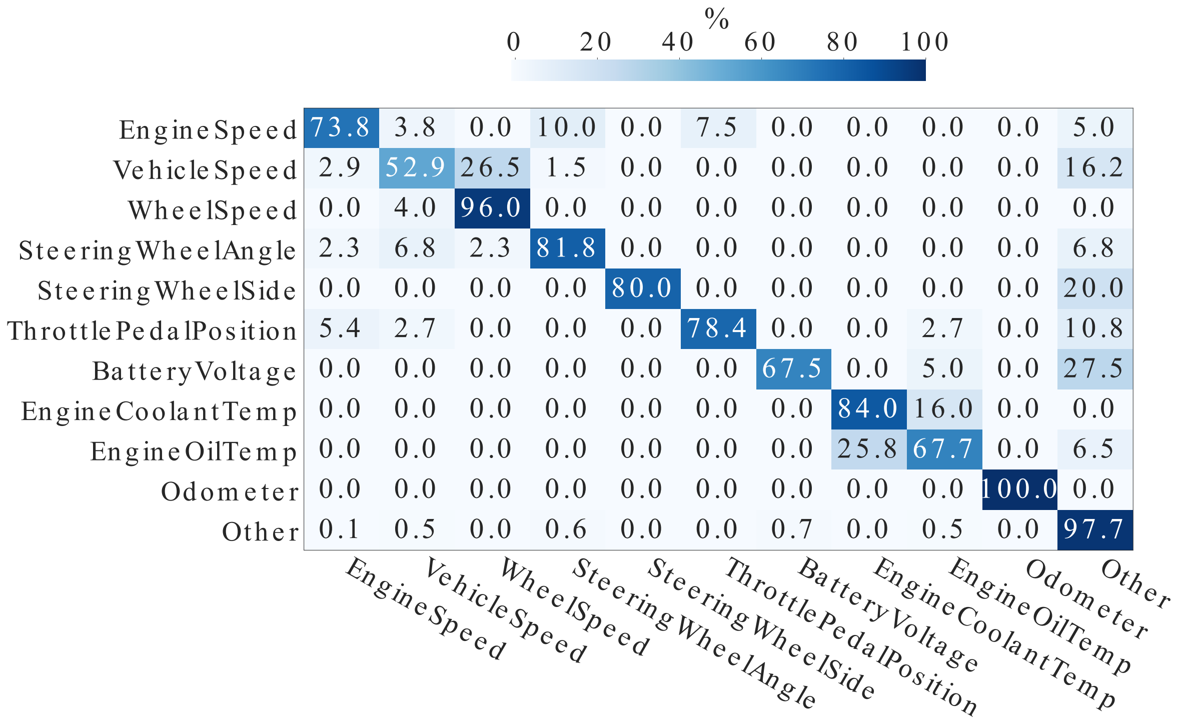
the different test sets. We compare the performance of CSI with the method proposed by Jaynes *et al.* [54], which we refer to as *Baseline*. To the best of our knowledge, this is the only related work that aims to translate the semantics of through supervised ML based on CAN data only.

Table 10 shows the *accuracy* and *F1-score* achieved by CSI with all the considered classifiers, as well as the results obtained with the Baseline method. For the latter, we show only the results achieved using RF, the best performing classifier in [54].

The accuracy is defined as the number of correct predictions divided by the number of total predictions made. The F1-Score is defined as the harmonic mean between precision and recall. Due to the presence of the background set, consisting of numerous signals, and to the fact that the foreground signals are present in different number in different vehicles, the test sets are imbalanced. As a consequence, the accuracy and F1-Score are influenced by the number of elements in each class given as input to the model. In this case, the results are heavily conditioned by the performance on the background set, as its number of samples is consistently higher than any other class. For this reason, we also report the balanced (or weighted) accuracy and F1-Score.

Our approach significantly outperforms the solution of Jaynes *et al.* [54] in all the considered metrics. The main reason for the poor performance achieved by their tool is that the samples are generated from all the bytes contained in the payload of a frame. The authors do not consider the possibility that a CAN frame can contain multiple signals. Therefore, the sample generation is negatively affected by the presence of unrelated chunks of payload that are associated to the signals.

Figure 24 illustrates the normalised confusion matrix, offering a complete view of the



	EngineSpeed	VehicleSpeed	WheelSpeed	SteeringWheelAngle	SteeringWheelSide	ThrottlePedalPosition	BatteryVoltage	EngineCoolantTemp	EngineOilTemp	Odometer	Other
EngineSpeed	73.8	3.8	0.0	10.0	0.0	7.5	0.0	0.0	0.0	0.0	5.0
VehicleSpeed	2.9	52.9	26.5	1.5	0.0	0.0	0.0	0.0	0.0	0.0	16.2
WheelSpeed	0.0	4.0	96.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
SteeringWheelAngle	2.3	6.8	2.3	81.8	0.0	0.0	0.0	0.0	0.0	0.0	6.8
SteeringWheelSide	0.0	0.0	0.0	0.0	80.0	0.0	0.0	0.0	0.0	0.0	20.0
ThrottlePedalPosition	5.4	2.7	0.0	0.0	0.0	78.4	0.0	0.0	2.7	0.0	10.8
BatteryVoltage	0.0	0.0	0.0	0.0	0.0	0.0	67.5	0.0	5.0	0.0	27.5
EngineCoolantTemp	0.0	0.0	0.0	0.0	0.0	0.0	0.0	84.0	16.0	0.0	0.0
EngineOilTemp	0.0	0.0	0.0	0.0	0.0	0.0	0.0	25.8	67.7	0.0	6.5
Odometer	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	100.0	0.0
Other	0.1	0.5	0.0	0.6	0.0	0.0	0.7	0.0	0.5	0.0	97.7

Figure 24: Confusion matrix of the predictions performed by CSI with MLP.

predictions made for each class by CSI with MLP, which is the best performing algorithm. The x-axis represents the predicted labels, while the y-axis the actual labels. The label "Other" corresponds to the background set. The main diagonal shows the rate of correctly classified samples for each class. Since the confusion matrix is normalised, the overall accuracy is calculated as the mean of these rates. The other slots report the rate of misclassification for each class (the rows being the false negatives and the columns the false positives).

The difference in the performance achieved by CSI for different signals can be explained by (i) the different quality of fitting provided by the features for each vehicle function and (ii) the presence of directly coupled signals. For instance, the Engine Oil Temperature is mostly misclassified with Engine Coolant Temperature and vice versa.

6.3 Discussion

In this section we presented CSI, an automated CAN translation algorithm that only requires raw CAN data as input. We validated the proposed solution on a dataset of real CAN traces collected from eight driving vehicles. Our solution is characterised by minimal hardware requirements. Using a combination of ML and ILP, CSI can automatically identify 10 safety-relevant vehicle functions on a pool of up to 59 signals with an accuracy up to 93 %.

However, the results obtained in this study are consistently limited by the assumptions related to the size of the background set and the redundant signals. Regarding the former, in this work we only considered a selected background set including signal for which we had the ground truth. In a real world scenario, the background set is likely to be composed by a significantly higher number of tokens. As highlighted by subsequent tests that we carried out, when considering a set of tokens extracted with a real tokeniser (see Section 4), CSI achieves poorly (mean accuracy for the Foreground (FG) set <30%). For what concerns the redundant signals, in principle it is not possible to know a priori the cardinality of each function in an unknown target vehicle (even though, it is still possible to set a plausible limit).

Future research should be conducted to extend the set of features to represent/fit better the vehicle functions taken into consideration in this work. In addition, to ensure a complete reverse engineering, to the translation of the semantic meaning of the signal should follow the translation of their format. Assuming no external ground truth related to the driving session, it is not clear how information related to the format could be inferred.

In conclusion, CSI showed that under limited assumptions it is possible to recognise critical functions carried by a number of signals. Additional work is needed to extend these findings and make this solution generalised and applicable in the real world.

7

Automated Reverse Engineering Based on Frame Matching

It is common knowledge that different car manufacturers equip their vehicles with ECUs from a limited number of Tier-1 suppliers, such as Robert Bosch GmbH [16] and DENSO Corporation [117]. It follows that an ECU can be part of the electronic system of multiple vehicle models. Intuitively, this is mostly true for models from the same brand or alliance, to enhance economies of scale. But, it can be also valid for models of different OEMs that get their supplies from the same Tier-1 supplier.

What does not seem to be public knowledge is that the CAN frames associated with these ECUs are also shared by different vehicle models. The direct implication of the reuse of CAN frames is that, by correctly reverse engineering signals within a frame with a certain ID in a vehicle model, we can assume that frames with the same ID found in other vehicles carry the same signals.

In this section, we present CANMatch, a framework for automated CAN bus reverse engineering that is able to decode the format of an unknown vehicle by exploiting frame ID re-utilisation. Differently from other proposed solutions – which require injection of diagnostic messages, use of external IMU/GPS data, mobile apps, expertise and partial manual effort – CANMatch gets in input only the CAN raw trace of the vehicle to reverse engineer and clear

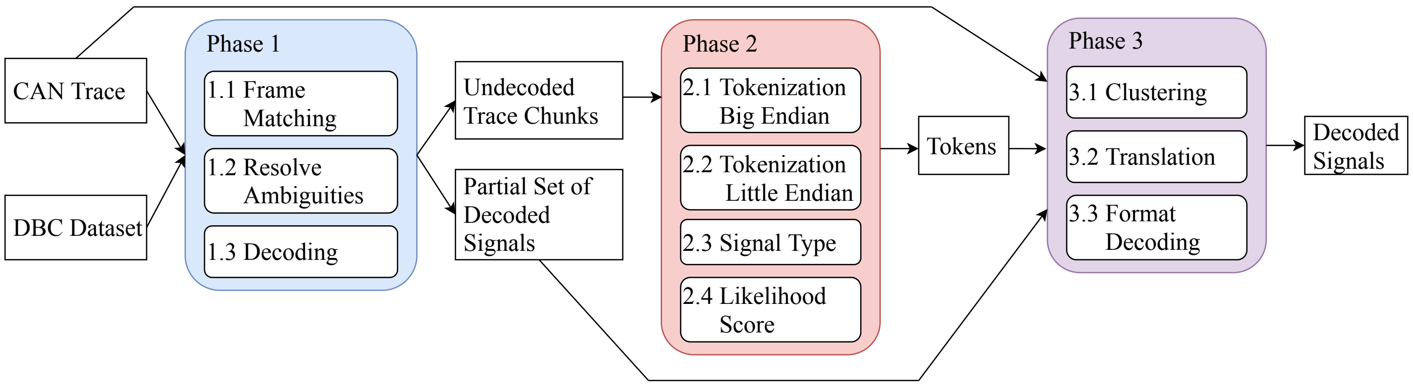


Figure 25: CANMatch pipeline: the framework takes in input a raw CAN trace and provides in output a list of decoded signals.

information on the formats adopted in other vehicle models. CANMatch aims to further speed up and scale the access to clear CAN data for all those researchers and aftermarket companies that drive the innovation in the automotive sector through data-enabled solutions.

7.1 Methodology

Provided that there is a ground truth dataset containing already decoded CAN traces from a number of vehicles (i.e., by using any of the methods described in Section 3.4), CANMatch decodes an unknown CAN trace by performing the following three main phases (illustrated in Figure 25):

- **Phase 1 (Frame Matching)** – CAN IDs and payloads are extracted, along with their timestamp. CANMatch searches for a match between the IDs found in the trace (to be decoded) and the same IDs present in the ground truth (previously decoded signals). If a match occurs, the frame is decoded according to the signals extracted from the matched frame in the ground truth dataset.

- **Phase 2 (Tokenisation)** – While part of the signals present in the trace have been decoded through Phase 1, portions of the trace are still not decoded (i.e. do which signals they contain is not known). The tokenisation task is performed on such unknown areas of the trace. The output is a set of tokens, represented by their boundaries and their endianness.

- **Phase 3 (Redundancy Resolution)** – Redundant signals are signals that carry the same vehicle function/telemetry. CANMatch looks for correlations between the signals decoded via Phase 1 and the tokens. If a match occurs, the vehicle function/telemetry of the known vehicle is attributed to the token. The scale factor and offset of the token are decoded too.

7.1.1 Phase 1

The goal of Phase 1 is to decode signals by matching the frames of the vehicle to reverse engineer with frames having the same ID, and extracting their content. The first step is to parse the CAN raw data into a standardised format and extract all the frame IDs. The second step is to query the ground truth dataset to find the same CAN IDs on previously reverse engineered vehicles. If all the frames associated with the same ID found in the ground truth have the same content, the matching is straightforward: the signals and all information related to them are assigned to the frame. This information includes the semantic meaning of the signals, their position within the frame, and their format (i.e. endianness, offset, scale factor and unit). In other words, the frame is decoded. An analysis of our ground truth set has revealed that 53.9 % of the frames associated to the same ID contain the same set of signals.

Vice versa, when different content is associated with the same frame ID for different vehicles in the ground truth, CANMatch needs to deal with the ambiguity. It does so by taking into account different properties of the frames. Specifically, all frames with a different length (as indicated by the DLC field) than the one currently analysed are preliminarily discarded. Intuitively, two frames with different lengths cannot contain the same set of signals. Then, a likelihood score is calculated taking into consideration the number of frame IDs in common between the target vehicle and the vehicles containing that same ID. The reasoning behind this approach is that vehicles with many IDs in common have higher probability to be equipped with the same ECUs and, therefore, to share the same signals (see Section 7.2.1). This is especially true for vehicle models produced by the same OEM. Finally, the frame associated with the highest likelihood score is considered as the reference frame. Similarly to direct frame matching, all the signals and their relative format found in the ground truth of the reference frame are attributed to the CAN ID of the target vehicle.

7.1.2 Phase 2

The trace chunks that are not decoded by Phase 1 (i.e., no matching frames are found) are processed in Phase 2 to identify the position of the rest of the signals through tokenisation. The tokenisation algorithm employed in this phase is the one presented in Section 4. However, some minor adjustments are necessary for frames that have been partially decoded, as the tokenisation should not performed on the whole payload but on chunks of it. Algorithm 2 illustrates the steps performed during Phase 2.

The input of Phase 2 is composed by the trace of the target vehicle I and the set of signals decoded through Phase 1 *PDS*. For each ID in the trace, a time series *ts* of its frames is extracted (lines 2-3). If no signal in *ts* was decoded through Phase 1, the tokenisation is performed on the whole payload as described in lines 4-12 of Algorithm 1 (lines 4-5). Instead, if one or more signals of *ts* were decoded through Phase 1, the undecoded chunks of the payload *ud_ts* are identified and a set of sub-time series is extracted accordingly (line 6). The decoded cross-byte signals of the current ID are then extracted (line 7). If there are no decoded cross-byte signals, the tokenisation is performed on the individual chunks in the way described in lines 4-12 of Algorithm 1 (lines 9-10). By contrast, if cross-byte signals were decoded through Phase 1, the endianness of the payload is extracted from them. If they are encoded according to the big endian format, *ud_ts* are tokenised according according to big endianness, otherwise according to little endianness (lines 11-17). Finally, the tokens, along with their type and endianness, are appended to the output T (line 20).

The reason behind the choice of considering only the endianness of the cross-byte signals, instead of all decoded signals, is motivated by the fact that, by default, generated DBC files indicate big endianness when the frame payloads have only in-byte signals. As a consequence, assuming big endianness without any validation based on cross-byte signals could lead to erroneous results.

Algorithm 2 Phase 2

```
Input: Input Trace $I$, Partial set of Decoded Signals $PDS$
Output: A set of Tokens $T$
 1: $T \leftarrow []$
 2: for $id$ in get_frame_IDS($I$) do
 3:     $ts \leftarrow$ get_frame_time_series($I$, $id$)
 4:     if partially_decoded($id$, $PDS$)==[] then
 5:         $tokens$, $types$, $endian \leftarrow$ tokenisation($ts$)
 6:     else
 7:         $ud\_ts \leftarrow$ extract_undecoded_chunks($ts$, $PDS$)
 8:         $cs\_signals \leftarrow$ cross_byte_signals($id$, $PDS$)
 9:         if $cs\_signals$==[] then
10:             $tokens$, $types$, $endian \leftarrow$ tokenisation($ud\_ts$)
11:         else
12:             if $endianness(cs\_signals) == 0$ then
13:                 $tokens$, $types \leftarrow$ tokenise_big_endian($ud\_ts$)
14:             else
15:                 $tokens$, $types \leftarrow$ tokenise_little_endian($ud\_ts$)
16:             end if
17:         end if
18:     end if
19: end for
20: $T$.append($<tokens, types, endian>$)
```

7.1.3 Phase 3

The goal of Phase 3 is to identify and decode signals among the tokens extracted by Phase 2 that bring the same vehicle functions as the signals decoded by Phase 1.

To assess whether two or more CAN signals are redundant, it is necessary to calculate a score based on the similarity of their time series, as extracted from the trace, and evaluate it against an acceptance threshold. In our case, the most straightforward approach would be a brute force one, i.e. calculate the score between each token and each signal decoded in Phase 1. Nonetheless, by definition, this approach is computationally inefficient. Phase 3 reduces the time needed to find these correlations by preliminarily clustering the time series of tokens and signals together. Then, the similarity score is only calculated between each token and the reference signal within the same cluster. In principle, just applying the clustering would be enough to identify the tokens that are redundant. However, as explained with more detail in Section 7.2.4, the clustering involves the risk of missing out the identification of some signals if not perfectly tuned. Once the vehicle function is identified, the token is fully decoded by calculating its format – scale factor and offset – through linear regression.

The pseudocode of Phase 3 is shown in Algorithm 3. Prior to the computation of the similarity scores, all the time series are interpolated in n points (line 3). The interpolation reduces the number of points in the time series and, therefore, contributes to a faster computation of the similarity scores. While the majority of CAN frames are sent with a periodicity between 10–50 ms, the information they carry is related to physical phenomena generated within the vehicle and, therefore, exhibit trends that can be encapsulated with one point per second granularity, which we consider is a good trade off between computational cost and information loss.

Redundant signals usually differ from one another in format. To minimise the impact of the scale in the computation of the similarity score, all time series are also scaled between 0 and 1 (line 4). Once the tokens and decoded signals are interpolated and scaled, the similarity score can be calculated between each of the tokens and each of the decoded signals within the same cluster (lines 5–13).

Algorithm 3 Phase 3

```
Input: Input Trace $I$, Partial set of Decoded Signals $PDS$, Tokens $T$
Output: $DS$ (Decoded Signals)
 1: $DS \leftarrow PDS$
 2: $T\_ts$, $PDS\_ts \leftarrow$ extract_time_series($I$, $T$, $PDS$)
 3: $T\_its$, $PDS\_its \leftarrow$ interpolate($I$, $T\_ts$, $PDS\_ts$)
 4: $T\_ists$, $PDS\_ists \leftarrow$ scale($I$, $T\_its$, $PDS\_its$)
 5: $C \leftarrow$ clustering($T\_ists$, $PDS\_ists$)
 6: for $c$ in $C$ do
 7:     $CS \leftarrow c$.get_signals()
 8:     for $t$ in $c$.get_tokens() do
 9:         $vf$, $mss \leftarrow$ translation($T\_ists$.get(t), $T\_ists$.get(CS))
10:         $f \leftarrow$ format_decoding($T\_its$.get(t), $T\_its$.get(mss))
11:         $DS$.append($<t$, $vf$, $f>$)
12:     end for
13: end for
14: $S \leftarrow$ get_speed_signals($DS$)
15: for any $s_i$, $s_j$ in $S$ do
16:     if $s_i$.get_frame_id()==$s_j$.get_frame_id() then
17:         $DS$.update_vehicle_function($s_i$,'WheelSpeed')
18:     else
19:         $DS$.update_vehicle_function($s_i$,'VehicleSpeed')
20:     end if
21: end for
```

It may happen that a token is found to be similar to more than one decoded signal. This is the case if the decoded signals are semantically related or if the threshold value θ is set too high. When such an ambiguity occurs, the token is attributed to the vehicle function of the decoded signal with the best (lowest) score (line 9).

Finally, Phase 3 calculates the scale factor and offset of the translated token (line 10). This is achieved by feeding a linear regression with the time series of the token and the time series of the redundant signal of reference. The slope of the function outputted by the linear regressor corresponds to the scale factor, while the intercept corresponds to the offset. Decoding the format of CAN dynamic signals by using linear regression was firstly introduced in [23]. In their work, the reference signals are the IMU and OBD-II data collected along with the CAN traffic and can be passed to the linear regressor as their values are directly interpretable by humans. In our case, instead, prior to the regression, the raw values of the reference signal have to be parsed to the human-readable format using the factor and offset extracted in Phase 1. As mentioned in Section 2.2, signals can have null factor (i.e. equal to 1) and null offset (i.e. equal to 0). According to the analysis on our ground truth set on SDDBC (see Section 2.5), 9.7% of the dynamic signals have such a format. For these signals there is no need to conduct the preliminary parsing.

To be noted that signals representing the wheels speed are often mislabelled as vehicle speed and vice versa. This is due to the intrinsic semantic correlation between these telemetries. The speed of the wheels can be distinguished apart because they are typically twinned. Phase 3 post-processes the results of the translation to exploit this peculiarity and, therefore, increase the overall accuracy (line 14–21). Vice versa, in the scope of the format decoding, each signal with a strong correlation to the vehicle speed can be used to decode the others. For instance, the front right wheel speed can be used as reference signal to decode the format of other wheels and vehicle speed.

7.2 Performance Evaluation

We first provide a preliminary data analysis of the considered dataset and describe the data collection process. Then, we present the performance of Phase 1 and Phase 3, evaluated

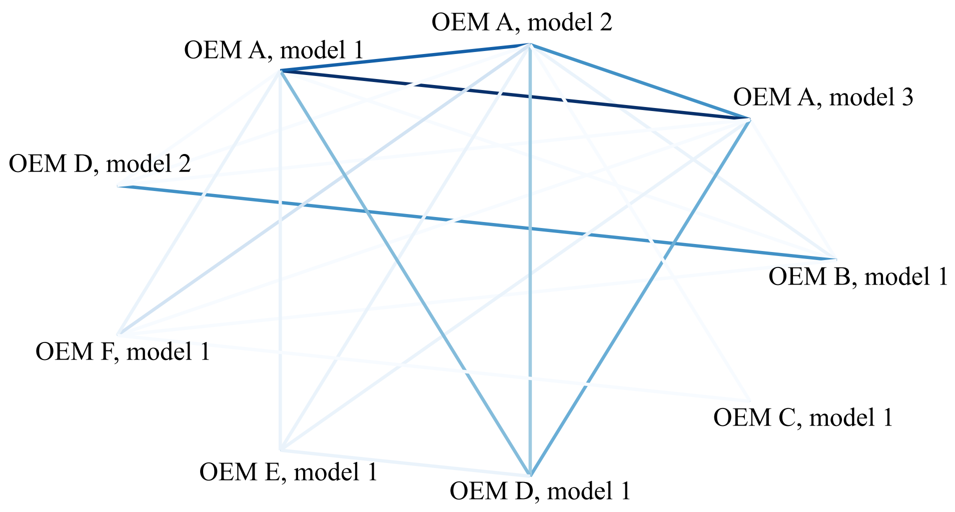


Figure 26: Example of reuse of CAN frames among vehicles from different constructors. The thicker the line, the more CAN IDs are shared between the two models.

independently (i.e., the evaluation of each phase does not depend on the evaluation results of the previous phase). For the evaluation of Phase 2 please refer to Section 4.3. Finally, we evaluate the performance of the entire CANMatch framework where all the phases are linked together. All the running times reported in this section refer to executions performed with a Dell Latitude 5490 laptop, equipped with Intel(R) Core(TM) i5-8250U CPU @ 1.60 GHz, 1800 MHz, 4 Core(s) with 8 Logical Processor(s).

7.2.1 Preliminary Data Analysis

From a preliminary analysis of SDDBC, it seems that a majority of the CAN IDs are reused in different vehicles, and so are the signals carried in their payloads. Specifically, we discovered that, out of 588 total frame IDs in the dataset, only 151 are uniquely found in only one vehicle, while all the others can be found in two or more vehicles.

To visualise effectively the relations between the different vehicles in terms of common frame IDs, we represent our dataset via a relation graph. A reduced version of this graph generated with nine (anonymised) vehicles only is illustrated in Figure 26. The nodes of this graph represent the vehicle models. There is an edge between any two nodes in the graph if the corresponding vehicles share at least one CAN frame ID. The edge thickness represents a weight corresponding to the number of common CAN frame IDs between two vehicle models. The full graph contains a total of 477 nodes and 8683 edges (not shown for readability reasons).

7.2.2 Data Collection

For the evaluation of Phase 1 and Phase 3, we employ the same ground truth used for the tokenisation (see Section 4.3.1). As raw CAN traces to be decoded, apart from DVT and ODVT, we also test Phase 1 on Parked Vehicles Traces (PVT) – a set of 477 CAN 10 s long raw traces and collected on vehicles being in idle mode, i.e., parked vehicles with no human action being performed. The ground truth of each of these vehicles is contained in SDDBC.

The reason why only Phase 1 was tested on PVT and not DVT and ODVT is because all event-driven signals related to a moving vehicle do not display any change over time when the vehicle is parked and, therefore cannot be detected by the tokenisation algorithm. Similarly, the time series of these signals do not present distinctive features that can be used in the clustering and translation tasks performed by Phase 3. As a consequence, we use DVT and ODVT for the validation of Phase 2 and Phase 3, as well as the entire pipeline. All traces in PVT are extracted with a `PCAN-USB FD` dongle.

Finally, all tests presented in the following sections are performed under real-world conditions, but the results refer only to the decoded signals present in the ground truth.

7.2.3 Phase 1 Evaluation

We evaluate the performance of Phase 1 by testing each trace in PVT through a *leave-one-out-cross-validation* approach. In other words, each trace is iteratively considered as belonging to an unknown vehicle to reverse engineer, whose ground truth is removed from the DBC dataset.

We analyse two performance metrics, recall and precision. These metrics highlight two key aspects to take into consideration while evaluating CAN reverse engineering: how extensive is the set of outputted decoded signals (recall), and how reliable is this output (precision). Other metrics widely employed in literature, such as F-Score, can be derived from recall and precision.

To understand how the size of the ground truth dataset influences the performance of

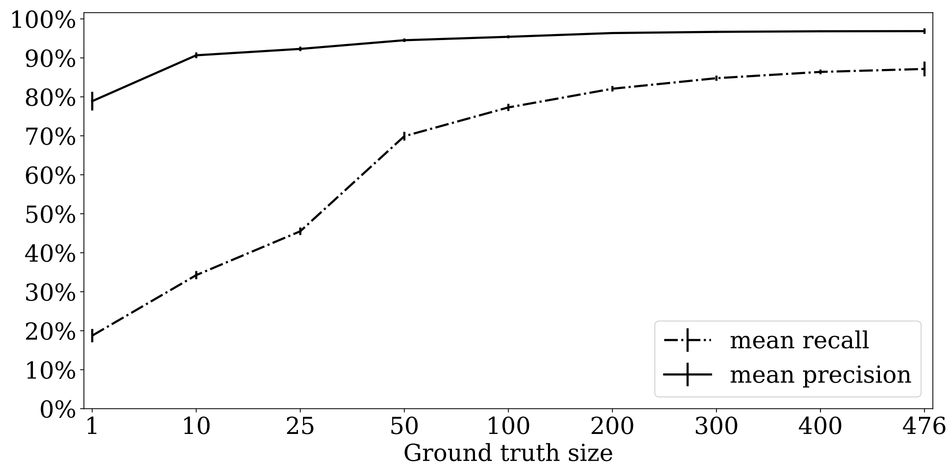


Figure 27: Mean recall and precision scored by CANMatch on datasets of different size.

Phase 1, we tested the algorithm on ground truth of different dimensions, starting from a ground truth composed of only one vehicle. The vehicles that constitute each subset are extracted randomly. To obtain results independent from the choice of the particular subset of vehicles, we perform this random selection 10 times for each ground truth size. It follows, that each vehicle in the PVT set is tested once when assuming the total ground truth, and 10 times when considering each of the inferior ground truth sizes.

Figure 27 illustrates the average values and 95 % confidence intervals obtained for all 477 vehicles and all iterations on ground truth size. With a ground truth set of only one vehicle, less than 20 % of the signals are correctly matched on average. The much higher precision (80 % circa) suggests that when a match occurs between a CAN ID in the target vehicle and a vehicle in the ground truth set, we can be fairly confident that the match is correct. As expected, both precision and recall increase with the dataset size. With a ground truth size comprising more than 25 vehicles, we can expect more than half of the signals to be decoded on average, with a precision superior to 90 %. Almost optimal recall and precision are achieved with a ground truth size of 200. After this threshold, the improvement in performance according to both metrics is marginal.

When considering all DBC files in our dataset (minus the one related to the current tested vehicle) as ground truth set, we obtain a mean recall of 83.3 % and a mean precision of 97.7 %.

The very high precision shows that almost all ambiguities are correctly solved. The lower recall is mostly due to the presence of unique frame IDs in the dataset. If a tested vehicle has one or more frames with an ID not present in the ground truth set, the signals within that frame cannot be decoded. Finally, it is to be noted that the frame matching provides the exact scale factor and offset for the given signals. It follows that the format decoding results can be directly inferred from the aforementioned results.

7.2.4 Phase 3 Evaluation

We evaluate Phase 3 independently of Phase 1 and Phase 2. We extract the time series related to each signal present in every trace of DVT and ODVT according to the information contained in their DBC files.

The tuning of the different parameters of Phase 3 (presented in the following) is based on DVT. Then, for each vehicle, we partition the time series into two distinct subsets: Decoded Signals (DS) and Tokens (T). The former set contains signals emulating the results of the decoding process of Phase 1, while the latter corresponds to the time series of the tokens that emulate the results of the tokenisation process of Phase 2. To analyse the performance for different number of signals decoded through Phase 1, we test Phase 3 assuming three different sizes of the DS set: 20, 50 and 80 %. As a consequence, we assume 80, 50 and 20 % for the size of T. We also partition the time series of DS and T randomly. To minimise the impact of this random choice on the evaluation, we repeat this selection 10 times.

The chosen metric for validating Phase 3 is the mean accuracy. In this case, a sample corresponds to a token in T to be translated. Let t be a token and R the set of all its redundant signals in T. We consider t correctly classified if:

- it is matched to any $s \in R$, if R is not empty;
- it is not matched to any signal in DS, if R is empty.

For this task, we consider three metrics that calculate the similarity between curves (time series): Area method, Root Mean Squared Error (RMSE) [118] and Dynamic Time Warping (DTW) [119]. The Area method measures the area of two curves in a 2-D space.

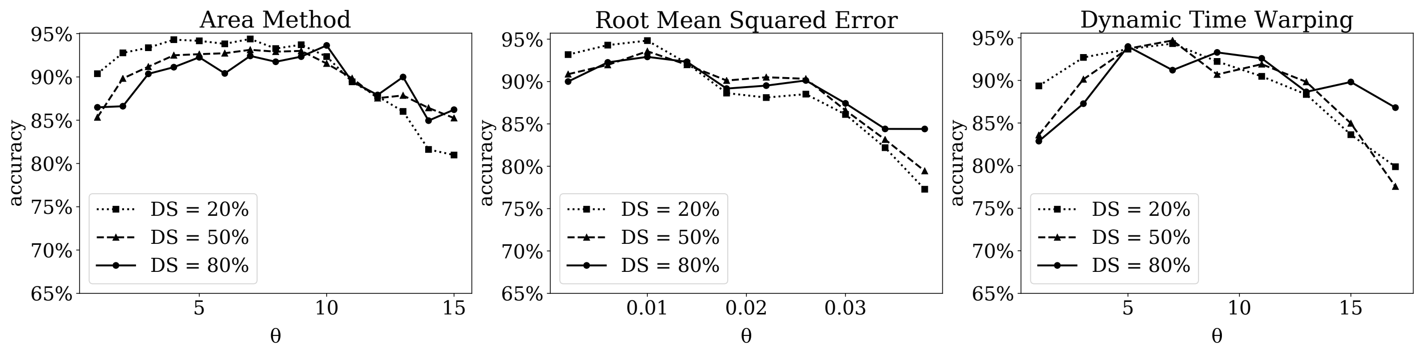


Figure 28: Tuning on the similarity threshold for Area method, RMSE and DFD at the varying of the size of *DS* size.

RMSE corresponds to the mean of the squared differences between each couple of points of two vectors for every x-coordinate. DTW finds an optimal correspondence between two time series, through a non-linear distortion with respect to the independent variable (here, the time).

The value of the threshold θ used to determine whether a token is similar enough to a signal is chosen through an extensive tuning taking into account the specific similarity metric used to calculate the score. Figure 28 illustrates how the accuracy of Phase 3 varies according to θ. All the results show that starting from the lowest value of θ, the accuracy increases until it reaches the optimal value, then it starts decreasing. In fact, if θ is set too low, some tokens are not matched with their redundant signal. If it is set too high, tokens with no corresponding redundant signal in *DS* are wrongly matched to another signal in *DS*.

The results also show that the properties of DTW, i.e., being resilient to temporal shifts and distortions, do not constitute an advantage in this case. In fact, the temporal skew between different redundant signal within the CAN bus is minimal, i.e., in the order of ms. On the contrary, DTW is sensitively more time-costly compared to the other two metrics. According to this tuning, a maximum accuracy of 94.8, 93.6 and 92.9 % is achieved with RMSE with $\theta = 0.01$ on a *DS* set of, respectively, 20, 50 and 80 % of the ground truth.

7.2.5 Clustering evaluation

The next step is to understand if performing clustering first and then computing the similarity scores within each cluster can help decreasing the computation time without decreasing the accuracy. For the selection of the clustering algorithm, we consider the following properties: i) it should not require to specify in input the number of clusters. In fact, the number of vehicle functions that can be found in a specific vehicle is not known a priori, ii) it should not require to specify in input the spatial distribution of the data, as it is not known a priori, iii) it should handle noise, i.e. outliers, because not all tokens are redundant of known signals (this property also helps to further reduce the computational cost, as outliers are excluded from the computation of the similarity scores), and iv) the time complexity should be as low as possible.

After taking into account a variety of clustering algorithms (e.g. centroid-based, hierarchical, distribution-based), we found out that density-based ones fit the aforementioned requirements. In particular, we focus on two most known density-based algorithms: DBSCAN [109] and Ordering Points To Identify the Clustering Structure (OPTICS) [120]. In this work, we chose the implementation of these algorithms offered by Scikit-learn [121], [122].

The hyperparameters of DBSCAN are ϵ, which corresponds to the maximum distance between two points to consider them neighbours, and *minPoints*, which corresponds to the number of neighbour samples to form a cluster. DBSCAN can find arbitrarily-shaped clusters of dense points, i.e., the cluster size is at least *minPoints*. Samples which lie in scarcely-dense areas are labelled as outliers. OPTICS is based on DBSCAN and uses the same parameters ϵ and *minPoints*. However, in case of OPTICS, ϵ can be ignored (or set to ∞), since it does not have any effect on the clustering accuracy [120].

In the following, we evaluate the impact of clustering on the vehicle function performance, by assuming RMSE as metric, with $\theta = 0.01$ and $DS = 50\,\%$. We set the parameter $minPoints = 2$. This is the minimum number of neighbour samples to form a cluster composed of at least one known signal and a token. We then tune ϵ for DBSCAN. The parameter ϵ should be set in a way that the outputted clusters i) should fit well to the vehicle functions in order to reduce the computational time, i.e. ideally there should be a ground truth signal representing only one vehicle function in each cluster, and ii) should be large enough so that all tokens representing the same vehicle functions are located in the same cluster.

Figure 29 illustrates three main metrics when varying ϵ: (i) mean number of clusters, (ii)

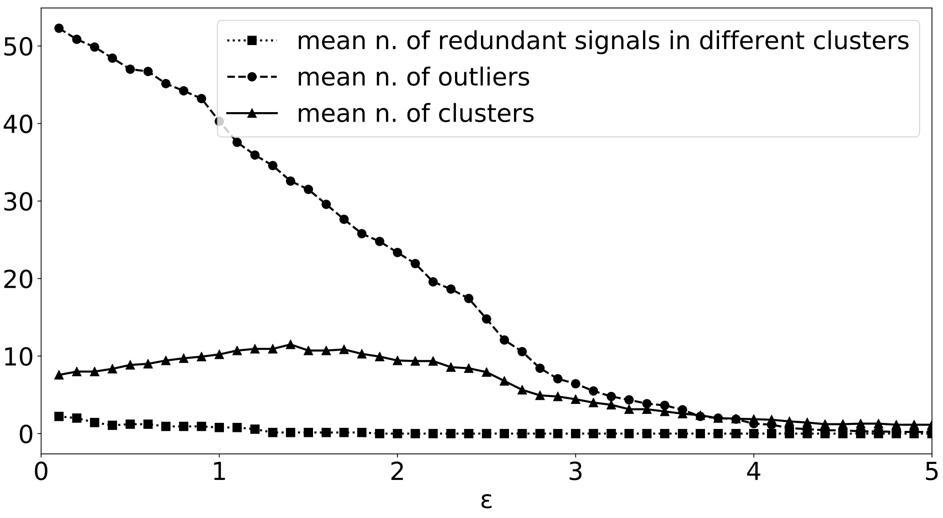


Figure 29: Tuning of parameter ϵ for DBSCAN.

mean number of outliers, and (iii) mean number of redundant signals in different clusters. It can be observed that the smaller the ϵ, the more samples are considered as noise (outliers). Also, the number of clusters initially increases with ϵ, as fewer samples are considered outliers, thus allowing the forming of new clusters. This growth is then followed by a decline, as the clusters become larger and incorporate samples previously belonging to multiple clusters. The figure also reports the number of redundant signals that are assigned to different clusters. As expected, when clusters are smaller on average, it happens more frequently that redundant signals are split into different clusters. The results reported in Figure 30 reflect the remarks made for Figure 29. For low values of ϵ, the mean accuracy of DBSCAN is inferior to the maximum value achieved without clustering. Vice versa, when the clusters are large enough, all the similarity scores relevant to identify all the redundant tokens are calculated.

Figure 31 compares the total execution time (ET), defined as the sum of time required for clustering (if performed) and time for the computation of all similarity scores, of DBSCAN, OPTICS and without preliminary clustering. The figure highlights that ET increases with ϵ in case of DBSCAN. In fact, the bigger the clusters are on average, the more similarity scores have to be calculated. For low values of ϵ (≤ 2), the ET using DBSCAN and OPTICS is one order of magnitude inferior compared to the baseline approach without clustering, while for

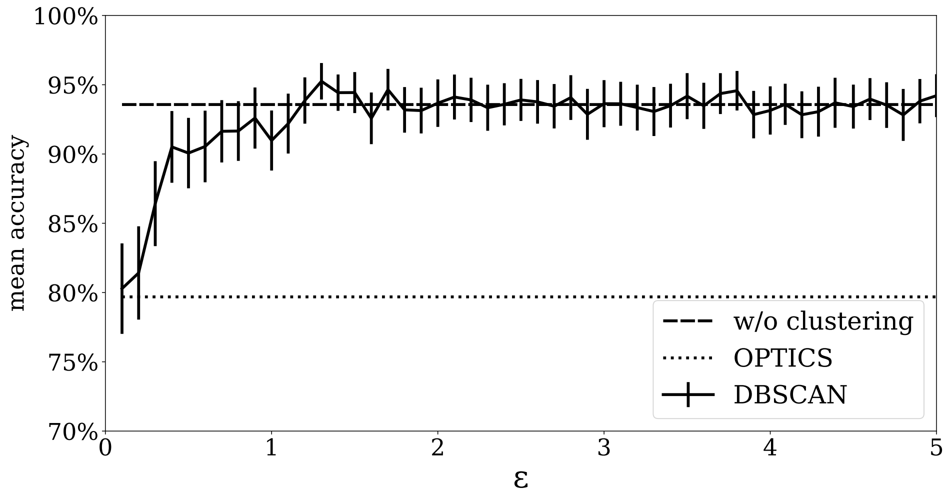


Figure 30: Comparison of the accuracy obtained for the translation of redundant signals between preliminary clustering through DBSCAN (tuned on ϵ), OPTICS and without clustering.

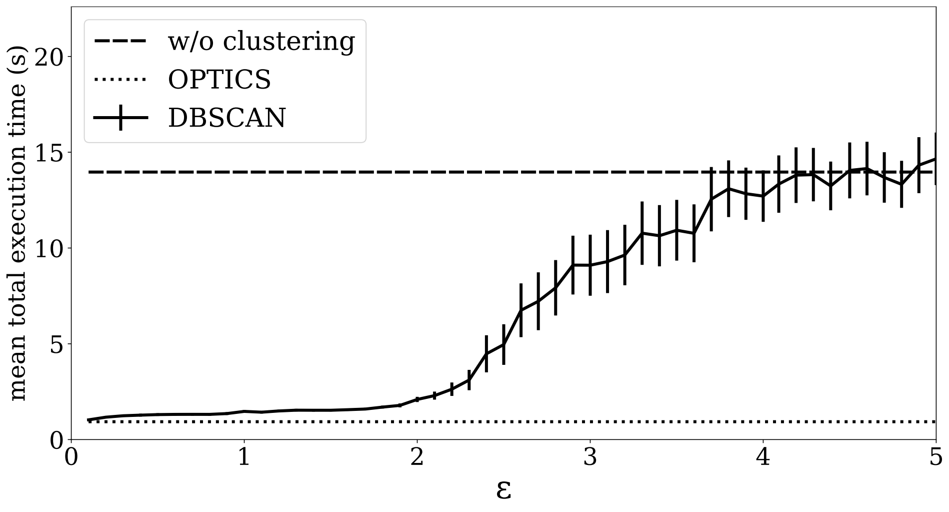


Figure 31: Comparison of the total execution time (ET) for the translation of redundant signals between preliminary clustering through DBSCAN (tuned on ϵ), OPTICS and without clustering.

high ϵ (≥ 4), ET is equal or higher.

Taking into consideration the results shown in Figures 30 and 31, we conclude that the optimal trade off between performance and total execution time for Phase 3 is obtained by performing clustering with DBSCAN and $1 < \epsilon < 2$.

7.2.6 Format decoding

After identifying the vehicle function of each redundant token, the scale factor and offset are decoded through linear regression, in the way described in Section 7.1.3. Let S_c be the time series of the target signal S parsed according to the calculated format, and S_o the time series of S parsed according to the original format, as defined in its ground truth. Let NRMSE be the RMSE between S_c and S_o, normalised by the range of values in S_o:

$$NRMSE = \frac{\mathrm{RMSE}(S_c, S_o)}{\max(S_o) - \min(S_o)} \tag{15}$$

We evaluate the accuracy format extracted for S through Format Decoding Accuracy (FDA), defined as:

$$FDA = 1 - NRMSE \tag{16}$$

We evaluate the FDA of the redundant signals of each vehicle in DVT and ODVT through a leave-one-out-cross-validation approach. Each signal is iteratively considered as the signal of reference, and all its redundant signals as the tokens whose vehicle function has been decoded through the similarity score and whose format is still unknown. Figure 32 shows the mean FDA obtained for each vehicle in DVT and ODVT. The figure shows that the performance of Phase 3 is consistent among different vehicles and different OEMs.

We also evaluate the accuracy of the format decoding by vehicle function, as reported in Figure 33. The presented vehicle functions are those for which redundant signals are present in the ground truth of SDDBC and OpenDBC. As mentioned in Section 7.1.3, the *speed* group is composed of the vehicle speed and the wheel speed signals. Figure 33 highlights that there is a moderate variability in the performance of Phase 3 on different vehicle functions.

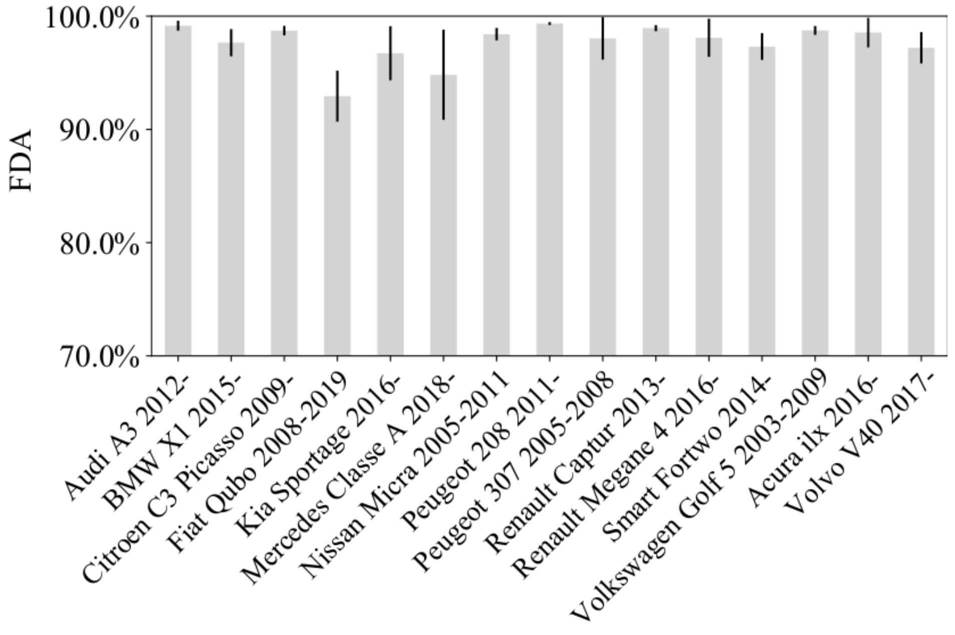


Figure 32: Phase 3 performance evaluation: mean FDA for each vehicle in DVT and ODVT.

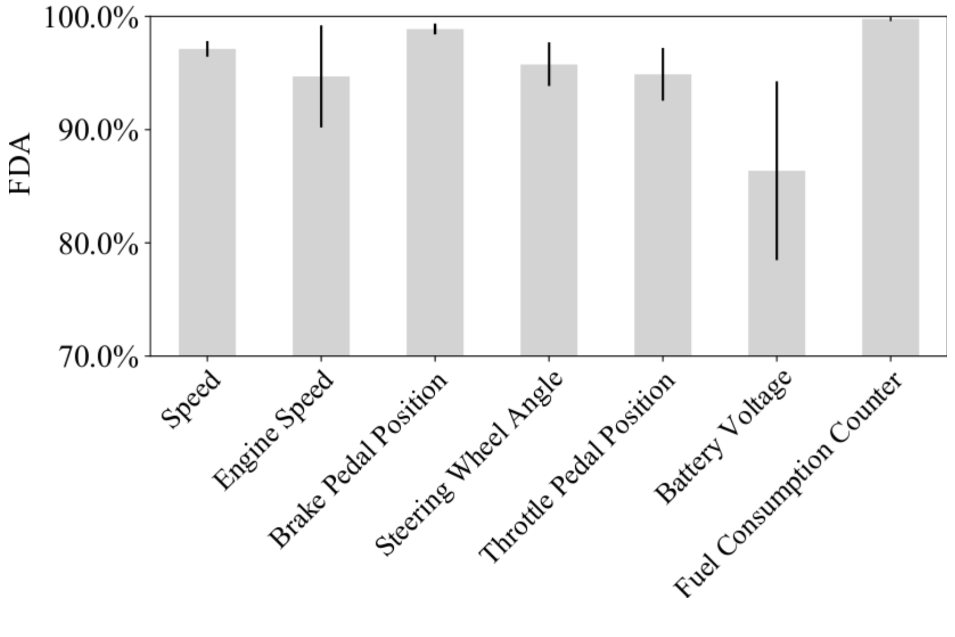


Figure 33: Phase 3 performance evaluation: mean FDA for each vehicle function represented by redundant signals in DVT and ODVT.

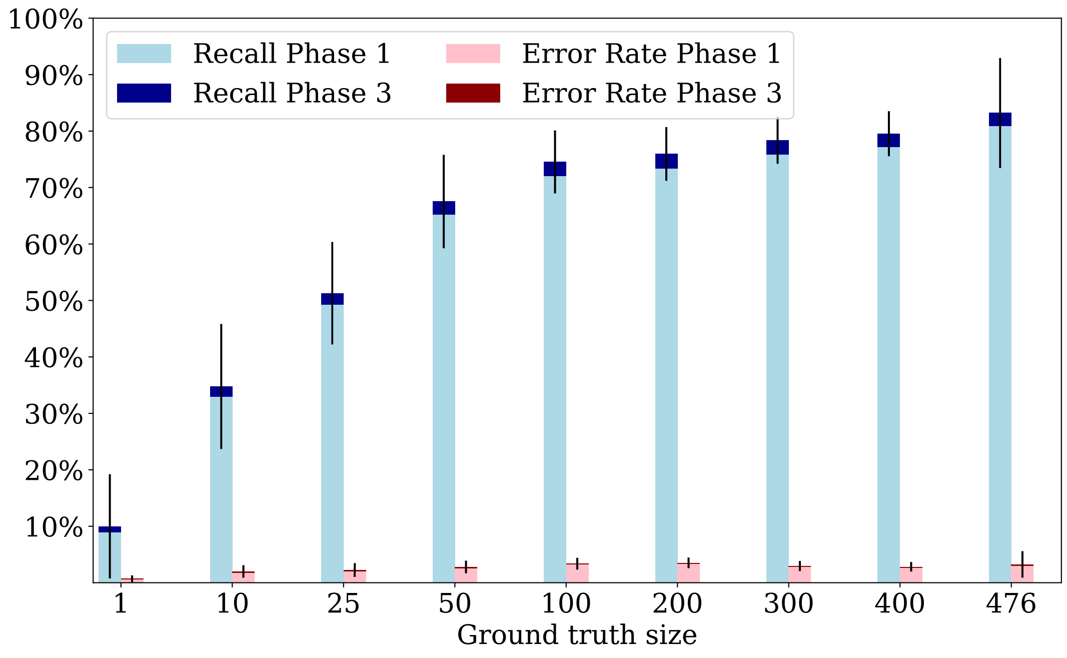


Figure 34: Recall and error rate for different sizes of the ground truth dataset.

7.2.7 CANMatch Framework Evaluation

We validate CANMatch using the DVT and ODVT sets and test the whole pipeline assuming different sizes of SDDBC.

Similarly to Phase 1, the vehicles composing each subset are chosen randomly. This selection is performed 10 times, to minimise the impact of this random choice on the overall evaluation. Each vehicle in the DVT and ODVT sets is tested once when assuming all DBC in our possession (minus the one currently under examination) in the ground truth set, and 10 times when considering a ground truth of smaller size.

Figure 34 illustrates the aggregated performance obtained for each ground truth size. We choose two metrics that show the completeness and the reliability of the translation of CAN signals achieved by CANMatch. In particular, since we want to highlight the cumulative contribution of each translation phase (i.e., 1 and 3) to the reverse engineering process, we employ the error rate metric (i.e., rate of incorrectly translated signals) instead of precision. The figure highlights that the majority of signals are decoded through Phase 1. As only

Table 11: Mean execution time for the different steps of the pipeline according for each vehicle in the DVT set.

Vehicle	**Time (s)**						
	Phase 1		**Phase 2**	**Phase 3**			**Total**
	Trace Parsing	Matching	Tokenisation	Clustering	Translation	Format Decoding	
Audi A3 2012-	5.1	12.5	24.8	1.2	4.0	0.1	47.6
BMW X1 2015-	3.3	14.4	24.7	0.9	9.8	0.1	53.2
Citroen C3 Picasso 2009-	5.3	5.8	4.5	0.2	1.1	0.1	17.1
Fiat Qubo 2008-2019	5.1	3.1	5.1	0.2	0.7	0.1	14.3
Kia Sportage 2016-	4.3	9.6	14.7	0.9	3.7	0.1	33.3
Mercedes A-Class 2018-	3.7	15.0	20.6	0.9	8.9	0.1	49.1
Nissan Micra 2005-2011	4.6	4.7	4.8	0.2	0.9	0.1	15.3
Peugeot 208 2011-	9.2	9.4	11.7	0.4	1.8	0.1	32.6
Peugeot 307 2005-2008	2.2	6.0	4.9	0.3	1.1	0.1	14.5
Renault Captur 2013-	12.3	7.5	13.0	0.4	1.8	0.1	35.1
Renault Megane 4 2016-	5.1	7.9	13.7	0.5	1.5	0.1	28.7
Smart Fortwo 2014-	11.3	8.9	17.7	0.4	1.3	0.1	39.7
VW Golf 5 2003-2009	3.0	8.1	12.3	0.7	2.8	0.1	27.0
Acura ILX 2013-	5.2	6.0	10.7	0.6	1.4	0.1	24.0
Volvo V40 2017-	4.5	9.2	12.0	0.8	1.6	0.1	28.2
Mean	5.6	8.6	13.0	0.5	2.8	0.1	30.6

13.1 % of the signals in the DVT are redundant, and given that most of them are also decoded through Phase 1, only few of them are left to be decoded by Phase 3. The figure also shows that the overall rate of signals which are wrongly decoded is almost irrelevant (especially those obtained through Phase 3), confirming the high precision of CANMatch.

It is also worth noting that the confidence interval is maximum when the ground truth includes only one vehicle and shrinks with the increase of the ground truth size, with the exception of the whole ground truth, for which the confidence interval is affected by the lower number of tests conducted. This confirms that the more comprehensive data we have in our ground truth set, in terms of quantity and diversity, the more reliable CANMatch becomes.

Table 11 reports the mean total execution time required by each step of the pipeline for each vehicle trace. The table shows high variance in the execution time among the different vehicles. The factors that seem to affect the computational load seem to be the year of release of the vehicle and the market segment. Indeed, latest and/or high-end vehicle models are usually equipped with more ECUs compared to older and/or cheaper ones. More ECUs means more frames to decode, hence longer time needed for the reverse engineering.

With a total computational time well below one minute for most of the tested vehicles (obtained with modest hardware resources), we conclude that, at the time of writing, CANMatch is the fastest tool for CAN bus reverse engineering.

7.3 Discussion

In this section, we presented CANMatch – an automated CAN bus reverse engineering framework that exploits the reuse of CAN frame IDs among vehicle models. To the best of our knowledge, CANMatch is the least-invasive and minimal approach in terms of employed hardware, manual effort, and execution time on the vehicle to reverse engineer. It requires in input the raw CAN trace to be decoded and a ground truth dataset of DBC files, necessary for the identification and decoding of signals in the initial phase. In addition, CANMatch proposes a method to identify redundant signals by exploiting a combination of density-based clustering and computation of similarities between tokens and previously-decoded signals.

We validated our solution on a diverse dataset of real CAN traces collected from 479 parked vehicles and 15 moving vehicles, obtaining comparable results on all of them, which advocates for its universal usage.

CANMatch offers a plug-and-play solution that, with minimum hardware equipment (a CAN dongle) and minimum amount of data (1-2 minutes of CAN raw trace) related to an unknown vehicle, is able to reverse engineer it in few seconds. However, our tool makes intensive use of DBC files which, if not acquired from third parties, implies that a (manual) reverse engineering process has to be previously put in place for a number of vehicles. While the data collection for the ground truth is certainly time consuming, the more vehicles are reverse engineered, the more accurate the process becomes. For these reasons, CANMatch is consistently more scalable than other tools for automated CAN bus reverse engineering. A combined use of our tool with other automated solutions would guarantee the fastest reverse engineering overall. A number of vehicles can be initially reverse engineered with semi-automated tools, such as LibreCAN or CAN-D [23], [24], in shorter time compared to the manual approach, until the ground truth is large enough to ensure the desired performance with CANMatch, which can then be used to significantly decrease time and effort of the overall reverse engineering process.

Future work includes additional optimisation for each step of the CANMatch pipeline to further reduce the overall computation time and increase accuracy. In addition, we intend to conduct a more comprehensive investigation upon the CAN format design choices for individual OEMs (e.g. priority and sending frequency of the frames). The goal is to discover hidden patterns that can be exploited to identify the sending ECUs and to use this information to decode the content of the frames, thus reducing the need for DBC files.

8

Benchmarking

The solutions that we proposed in Sections 5 to 7, as well as the tools that were presented in the literature (see Section 3), are highly diverse in terms of followed approach (see Section 8), requirements – such as software, hardware, and human effort – testbeds, and evaluation metrics. The majority of these projects have been developed in collaboration with industrial partners, with the consequence that they are not open-source. Technical details are also often protected by non-disclosure agreements, which make the reproducibility a complicated task.

The analysed work focuses on retrieving different information regarding the CAN signals. For example, some studies focus exclusively on finding the boundaries [27], [28], [107], while others only seek to identify the semantic meaning of the frames [32], [54], [106]. In addition, the varying granularity of the decoding process in terms of extracted properties makes the quantitative comparison of the solutions presented in this survey difficult. Hereafter, we summarise and compare the main characteristics of the presented methodologies.

Table 12 reports the types of tokens identified by the tokenisation algorithms and the approach followed to deal with the endianness. All methodologies are listed according to their approach. The table shows that most of the tokenisation algorithms are capable of identifying the majority of token types, while only few address the endianness.

Table 13 compares the translation methodologies. For each algorithm, the table outlines the requirements needed in terms of hardware, software and data, as well as the level of

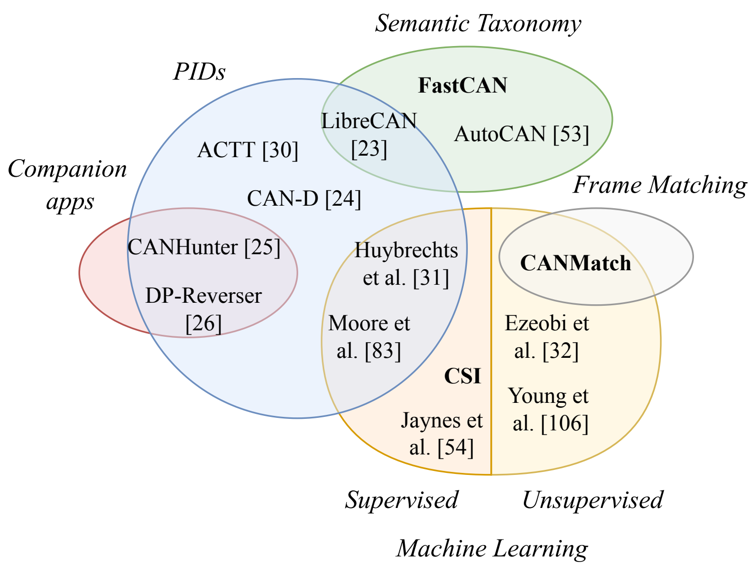


Figure 35: Classification of work on CAN bus reverse engineering.

Table 12: Comparison of the tokenisation algorithms

	Approach	**Endianness approach**	**Physical**	**Counter**	**Checksum**	**Status**	**Constant /Unused**
Markovitz and Wool [27]	Optimisation	–	✓	✓	–	✓	✓
ACTT [30]	Optimisation	Reverse bit ordering	✓	✓	✓	–	✓
READ [28]	BFR	–	✓	✓	✓	✓	✓
TANG [107]	BFR	Reverse bit ordering	✓	–	✓	–	–
LibreCAN [23]	BFR	–	✓	✓	✓	✓	✓
AutoCAN [53]	Optimisation	–	✓	✓	✓	✓	✓
CAN-D [24]	Optimisation	Conditional probability of non-consecutive bits flips	✓	✓	✓	✓	✓
CANMatch	BFR	Reverse bits within single bytes	✓	✓	✓	✓	✓
Choi *et al.* [29]	BFR	–	✓	✓	✓	✓	✓

Table 13: Comparison of the translation algorithms.

	Approach	**Requirements**	**Automation**	**Intrusiveness**
Jaynes *et al.* [54]	Supervised ML	DBC files and CAN logs of a number of vehicle models	Full	No
Huybrechts *et al.* [31]	Supervised ML, PIDs	GPS data, PIDs requests through the OBD-II port	Full	High
ACTT [30]	PIDs	PIDs requests through the OBD-II port	Full	High
LibreCAN [23]	Taxonomy, PIDs	GPS and IMU data, PIDs requests through the OBD-II port, instructions for a human operator regarding the data collection	Partial	High
Moore *et al.* [83]	Supervised ML, PIDs	PIDs requests through the OBD-II port	Partial	High
CSI	Supervised ML	DBC files and CAN logs of a number of vehicle models	Full	No
Ezeobi *et al.* [32]	Unsupervised ML	–	Full	No
Young et al. [106]	Unsupervised ML	–	Full	No
AutoCAN [53]	Taxonomy	GPS data	Full	Moderate
CANHunter [25]	Companion apps, PIDs	Mobile apps installed on a mobile phone	Full	High
CAN-D [24]	PIDs	PIDs requests through the OBD-II port	Full	High
CANMatch	Frame matching, unsupervised ML	DBC files of a large number of vehicle models	Full	No
FastCAN	Taxonomy	GPS data, instructions for a human operator regarding the data collection	Partial	Moderate
DP-Reverser [26]	Companion apps, PIDs	Companion apps, cameras, a robotic arm	Full	High

automation achieved and their degree of *intrusiveness*. The approaches on which each work is based are also recapped. It is to be noted that the CAN dongles and the CAN trace of the target vehicle are omitted from the requirements column because they are employed in all methodologies.

Intrusiveness refers to the active and voluntary alteration of the traffic transiting on the bus. The traffic can be manipulated with the injection of messages or with the triggering of target ECUs by a human operator. Intrusive approaches present two main limitations: (i) they typically require additional equipment that must be physically attached in the vehicle, such as a diagnostic dongle to install on the OBD-II port, and (ii) the necessity for a human operator implies the risk that they would not be able to follow precisely the instructions, thus compromising the quality of the collected data.

The table highlights that taxonomy-based methodologies are the only ones to achieve partial

Table 14: Properties decoded by CAN reverse engineering methodologies

	Boundaries	**Endianness**	**Signedness**	**Semantic Meaning**	**Format**
Jaynes *et al.* [54]	–	–	–	At frame level	–
Markovitz and Wool [27]	✓	–	–	–	–
Huybrechts *et al.* [31]	–	–	✓	–	
TANG [107]	✓	–	–	–	–
READ [28]	✓	–	–	–	–
ACTT [30]	–	–	✓	✓	
LibreCAN [23]	✓	–	–	✓	✓
Moore *et al.* [83]	✓	–	–	✓	–
CSI	–	–	–	✓	–
Ezeobi *et al.* [32]	–	–	–	At frame level	–
Young et al. [106]	–	–	–	At frame level	–
AutoCAN [53]	✓	–	✓	✓	
CANHunter [25]	✓	–	–	✓	–
CAN-D [24]	✓	✓	✓	✓	
CANMatch	✓	✓	–	✓	✓
FastCAN	✓ (CANMatch tokenizer)	✓	✓	✓	✓
DP-Reverser [26]	✓	–	–	✓	✓

automation, while the other approaches enable full automation of the reverse engineering process. However, Table 14, which reports all the properties decoded by each methodology, shows that taxonomy-based approach guarantees a more complete output compared to the majority of other approaches.

Part IV

PREVENTING REVERSE ENGINEERING

9 Introduction to CAN Bus Security

CAN data is heavily employed to enable academic research on vehicular technologies and drive innovation in the private sector. The progressive automation of CAN bus reverse engineering allows researchers and aftermarket companies to easily obtain CAN data from vehicles and develop innovative automotive services and use cases. However, this automation also raises significant concerns regarding the privacy and security of vehicles. Hereafter, we discuss the main security issues and the potential attacks on the CAN bus, as well as the countermeasures proposed for them. We contextualise these solutions in the field of reverse engineering with the aim of understanding whether they are sufficient to prevent it as well.

9.1 Attacks on CAN Bus

It has been shown in the literature that the signals transiting on the CAN bus can be effectively employed to fingerprint the drivers [10], [123]–[125]. This possibility can be exploited by governmental agencies, insurance companies, or potential adversaries to violate the privacy of the person behind the steering wheel. Moreover, an alarming number of physical and remote attacks posing a threat for the safety of vehicles and passengers have been demonstrated successful [10], [36], [126]–[130].

In 2015, Miller and Valasek [36] took over a Jeep Cherokee remotely and drove it off the road. They were able to take control of the steering wheel and brakes by injecting CAN messages from the compromised infotainment system. This attack shows that the absence of ECU authentication enables potential adversaries to replace the actual value carried by any CAN signal with malicious content.

Nevertheless, at the time of the attack, no work on the automation of CAN reverse engineering had been presented yet. As a consequence, the authors spent several months to achieve a precise understanding of the CAN encoding, which is necessary to carry on the attack exactly as it is designed. Additionally, such an attack requires the adversary to have constant access to a vehicle of the same model as the target vehicle during the whole reverse engineering process. Despite the fact that this attack caused initially public stir [131], the difficulty and time required for its preparation made the interest of OEMs towards CAN security quickly fade away. As a result, nowadays vehicles are not equipped with a secure version of the CAN bus yet.

The automation of CAN bus reverse engineering, however, is a game changer with respect to the preparation of vehicular attacks. The easing of this process will likely encourage potential adversaries to design a number of new attacks. Similarly, reducing the data collection time implies that the attacker only needs access to a vehicle of the same model as the target vehicle for only a short time span. Furthermore, recent advances in Vehicle-to-Everything (V2X) communication technologies and related use cases [42] are expected to boost the spread of CAVs [45]. As a consequence, we can expect the number of ECUs equipped with a wireless interface to rise consistently. From a security perspective, this means an increase of the attacking surface that an adversary can exploit to gain remote access to the CAN bus and inject malicious frames in a similar fashion as [36].

This scenario looks even worse if we factor in the massive reuse of ECUs across different vehicle models (as shown in Section 7). Some ECUs seem to be present in a large number of vehicles across different car brands. Assuming that in the near future some of the popular ECUs will be equipped with a wireless interface, an attacker would need to hack just few of them to gain remote access to the internal data of a high percentage of vehicles on the road. By adopting fully automated reverse engineering techniques, adversaries could also reverse engineer the CAN bus of unknown vehicle models without having physical access to them.

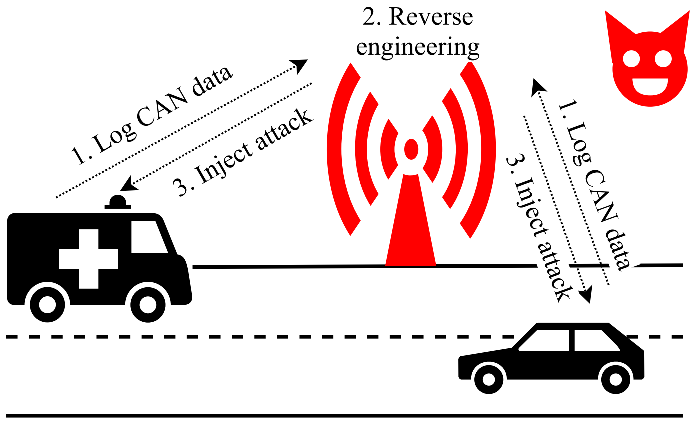


Figure 36: Remote attacking scenario: a compromised RSU is exploited to log CAN data from two vehicles nearby, reverse engineer the data, and inject a pre-designed vehicle-agnostic attack.

In such a scenario, they can potentially reverse engineer multiple vehicles and, subsequently, inject pre-designed vehicle-agnostic attacks in all of them in a single event. Figure 36 illustrates an attack scenario in which an adversary reverse engineers and injects tampered messages fully remotely through a compromised RSU in communication with vehicles on the road.

9.2 Countermeasures

Many solutions have been proposed in the literature to secure the CAN bus. A first straightforward approach is *network segmentation* [132]. Its goal is to limit the damage caused by attacks by splitting the CAN bus into distinct sub-networks. If a sub-network is compromised, the attack is contained and does not propagate to the rest of the network. A gateway handles the communication between sub-networks. This method is currently applied on a number of commercial vehicles, but it presents some evident limitations. Namely, if the gateway is compromised, the attack can be extended to the whole network. Additionally, this solution can increase the maintenance cost of the bus [133].

A theoretically more robust approach is the adoption of encryption mechanisms [134]–[139]. A number of cryptographic algorithms have been proposed at software and hardware level, such as Advanced Encryption Standard (AES)-128/256 [135], [136], Elliptic-curve Diffie–Hellman [136], and Triple Data Encryption Standard (DES) [134]. However, these techniques do not address *replay attacks* which, in the case of CAN bus, are non-negligible. Namely, an attacker can override the actual value of telemetries and other vehicle functions by continuously replaying the same messages, thus sabotaging the correct functioning of the vehicle.

As discussed in Section 9.1, a major issue in CAN security is the absence of ECU authentication. A multitude of approaches have been proposed in literature to enable ECUs to verify the source of the frames [140]–[144]. In addition, AUTOSAR built a module, Secure Onboard Communication (SecOC), based on ECU authentication to prevent tampering and spoofing. SecOC also guarantees a defence against replay attacks through counter-based freshness management [145]. Two main obstacles to achieve a large scale implementation of authentication systems for in-vehicle networks are the costs related to the increased computational requirements of the ECUs, and the traffic overhead.

The most investigated classes of countermeasure techniques against CAN bus are the Intrusion Detection Systems (IDSs). We can identify two categories of IDS: *signature-based* and *anomaly-based*. Signature-based systems establish a taxonomy of known attacks and scan the CAN traffic to find a match with any of them. The anomaly-based systems, instead, observe the ECUs and CAN traffic in search of behaviours that deviate from the usual nature of the network. The methodologies for anomaly-based IDS can be grouped into the following categories: (i) physical characteristics-based [39], [41], [146] (ii) sending frequency-based [147]–[149] (iii) feature-based [150], [151], and (iv) specification-based [152], [153]. Anomaly-based IDS generally achieve lower detection performance compared to the signature-based ones but, differently from those, they are capable of detecting unknown attacks.

The popularity of IDS is due to the fact that they do not typically require substantial changes to the CAN protocol and add low or no overhead to the communication stream. While they have shown to be an efficient countermeasure against attacks on the CAN bus, they cannot prevent reverse engineering. In fact, the assumption of IDS is that adversaries have an *active* role in the attack, i.e. they actively inject tampered information. Similarly, authentication systems are not sufficient to prevent adversaries from sniffing the CAN traffic.

On the contrary, the logging of data to perform automated reverse engineering is a *passive* task that does not alter the regular traffic. Hence, IDSs are not able to detect suspicious activities. In principle, it would be possible to adapt IDS to recognise the abnormal request of PIDs, as it implies an alteration of the traffic. However, diagnostics requests are a useful tool employed by car electricians in their daily job. We argue that it is unlikely that a tool would manage to distinguish between the usage of PIDs for diagnostic purposes and for reverse engineering, since the type and timing of the requests is highly arbitrary. Moreover, since most of PIDs address emissions-related parameters, their usage could become limited with the advent of EVs.

A suitable approach to prevent reverse engineering is the addition of an encryption layer to the frame payloads. In fact, it is virtually impossible to find the boundaries of the signals and identify their semantic meaning and format in an encrypted payload. Unfortunately, solutions based on strong cryptography do not seem fitting the constraints of the ECUs of commercial cars in terms of memory and computational power [133]. Furthermore, the limited length of CAN standard payload hinders the encryption of CAN data. To overcome this limitation, the information could be split into different frames sent consecutively. However, this solution would imply substantial traffic overhead which, in safety-critical system such as CAN, is a major issue.

In conclusion, further research is needed to identify solutions compliant with the constraints of CAN bus. To gain the possibility of being adopted on a large scale by the OEMs, the proposed methodologies should not cause significant changes to the protocol and come at an inexpensive cost.

10 Preventing Frame Matching through IDs Anonymisation

In Section 7 we unveiled that the reuse of the IDs of CAN frames sent by ECUs can be exploited to carry out reverse engineering in a highly automated way. Differently from the majority of related work, this approach is event-agnostic, i.e. the data can be decoded without knowing the events occurred at data collection time. By using this method, an attacker with remote access to the CAN bus of a connected vehicle can perform reverse engineer and inject an attack without physical access nor prior knowledge about the target vehicle.

OEMs seem unwilling to bring core modifications to the CAN protocol (i.e., by adding encryption) as it would inevitably bring massive disruptions in the supply chains. One potential solution to protect against frame matching-based reverse engineering approaches and improve the security of the CAN bus, while meeting the necessities of the OEMs, is to avoid the reuse of frame IDs for newly released vehicle models.

In this section, we investigate whether the abandonment of the frame ID reuse practice is sufficient to completely anonymise the frames, thus nullifying the benefit of a reverse engineering approach based on frame matching. In particular, we study the possibility of performing matching on anonymised frames by exploiting other properties of the frames. If confirmed, the implication would be that frame-matching based algorithms can still be used to perform CAN bus reverse engineering despite the ID anonymisation with minimal extra effort.

This would further corroborate the view that OEMs must improve the CAN bus security by applying substantial modifications to the protocol.

To validate this thesis, we propose a frame deanonymisation algorithm, which makes use of ML models to fingerprint frames based on their characteristics and dynamic behaviour. The ML models can then recognise the frames in a new vehicle to reverse engineer based on previously acquired knowledge.

10.1 Methodology

In this work, we assume that frame IDs are anonymised, i.e., an ECU model manufactured by the same OEM and installed in any two different vehicle models use different frame IDs for frames carrying the same vehicle functions. In particular, we make two fundamental assumptions:

1. Following the standard CAN protocol, the new ID associated to a frame should uniquely identify that frame.
2. The attribution of new IDs operated by the OEM should not be random, but rather preserve the frames priority. This assumption is important because lower-priority frames have more chances to be overwritten by higher-priority frames, when sent on the CAN bus simultaneously.

Hereafter, we present a tool to deanonymise frames which are anonymised following the aforementioned assumptions. The pipeline of this method is shown in Figure 37. The tool firstly divides the CAN trace collected from the vehicle to reverse engineer into n sub-traces, where n corresponds to the amount of unique IDs. Every sub-trace contains the payload and DLC of the frames with the same ID, following the temporal order in which they appear in the trace. Then, from each sub-trace we extract six features described as follows:

- **Payload Length** – it corresponds to the number of data payload bytes, as reported in the DLC field.

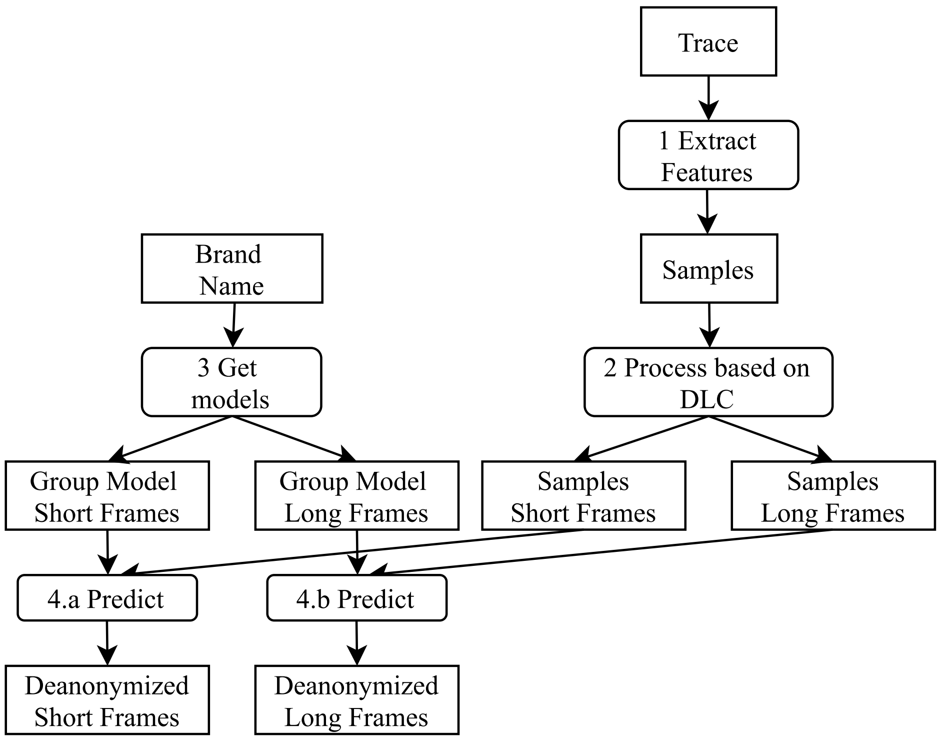


Figure 37: Pipeline of the proposed solution.

- **Mean Sending Frequency** – it is the mean frames sending frequency, as recorded by the CAN dongle.

- **Standard Deviation of Sending Frequency** – it is the standard deviation of the frames sending frequency, as recorded by the CAN dongle. The sending frequency is influenced by the precision of the internal clock of the sending ECU. As demonstrated by Cho and Shin [39], the difference between the taget period and the actual period can be used to fingerprint the sending ECUs. In this case, we use this property to fingerprint the frames.

- **Percentage of Active Bits** – it corresponds to the percentage of bits that flip at least once among the total number of bits in the frame payload throughout the trace.

- **Mean Bit Flip Rate** – it is the mean flip rate of the bits in the frame.

- **Frame Priority** – the frames within a trace are divided into quartiles based on their priority, which is given by their frame ID.

The trace splitting and features extraction process are referred in Figure 37 (step 1).

Once a sample is generated for every sub-trace in the described manner, the algorithm splits the set of samples into two distinct subsets according to their length (extracted from the DLC) (see step 2, Figure 37). The first subset contains all frames whose payload is 8 Byte while the other includes all frames whose payload is shorter than 8 Byte. We refer to frames in the former set as *long frames* and to the frames in the latter as *short frames*.

From a preliminary analysis of a dataset of 427 traces in PVT(see Section 7.2.1, we discovered that as much as 70.5 % of frames are long, while 29.5 % are short. The reason behind this division is that the feature Payload Length, while constituting a helpful discriminant for fingerprinting the short frames, is ineffective for the long frames. Therefore, this feature is removed from the set of long frames.

Each of the two sets of samples is given in input to an ML model, specifically trained on samples (see step 3, Figure 37) of the same kind (i.e., from long or short frames). This model is trained exclusively on samples from the same industrial group/alliance of the current vehicle to reverse engineer. The underlying rationale is that vehicles are more likely to have more frames in common with models from the same industrial group. Preliminary tests highlight that training specifically a classifier for each industrial group rather than on all samples leads to lower amount of misclassified samples, due to the reduced variance. Finally, all the samples are deanonymised (see step 4, Figure 37).

10.2 Performance Evaluation

Our algorithm is evaluated on a subset of PVT composed of 427 10 s CAN traces obtained from from 28 different automotive brands belonging to 15 different industrial groups from EU, USA, Japan, India and South Korea. In order to train the classifier with a higher number of samples, we generate five subtraces from each CAN trace through sliding window.

The vehicles in our evaluation set contain a total of 33 034 CAN frame IDs, from which we extract an equal number of samples. Table 15 summarises other relevant information for all (anonymised) industrial groups in the evaluation set. For each industrial group, the table

Table 15: Evaluation set statistics

OEM group	**n. vehicle traces**	**n. unique frames**	**mean n. frames per vehicle**	**long / short frames (%)**
A	4	65	32.75	61.1 / 39.9
B	5	87	45.4	58.6 / 41.4
C	6	119	57.5	47.0 / 53.0
D	6	193	52.5	88.9 / 11.1
E	10	164	53.6	95.0 / 5.0
F	14	289	47.1	99.7 / 0.3
G	24	190	33.5	82.3 / 17.7
H	25	335	73.28	86.1 / 13.9
I	25	323	71.2	77.1 / 22.9
J	25	293	68.7	49.9 / 50.1
K	32	299	43.9	96.2 / 3.8
L	33	491	34.5	71.4 / 28.6
M	60	548	40.4	45.8 / 54.2
N	76	502	40.2	56.0 / 44
O	82	630	55.2	83.3 / 16.7

reports:

- The number of vehicle traces (n. vehicle traces).
- The number of unique frames that are found across all vehicle models (n. unique frames), as identified by their ID. The table highlights that a bigger and more variegated set of vehicle models corresponds to an increase in the total number of unique frames.
- The mean number of unique frames per vehicle (mean n. frames per vehicle). It depends on the level of digitalisation of the vehicles produced by the manufacturer, which is usually correlated to the market segment (i.e. high-end vehicle models have more electronic components).
- The percentage of long and short frames on the total number of unique frames (long / short frames).

10.3 Performance Metrics

In our evaluation we adopt a leave-one-out-cross-validation approach. Namely, each vehicle in the dataset is iteratively considered as the vehicle to reverse engineer and its ground truth is discarded. The classifiers are then trained on the rest of the dataset.

Our classification task is an Open Set Recognition (OSR) problem. In an OSR problem the knowledge of the world is incomplete at training time [154]. As opposed to the standard closed-world classification scenario, the ML models do not have only to classify samples from known classes, but also to adequately reject samples belonging to unknown classes. As a matter of fact, apart from the ECUs mounted in previous vehicle models, we expect newly released vehicle models to be also equipped with last generation components, which will send frames unseen until then. For this reason, the trace of a new vehicle to reverse engineer may contain frames that the ML model has never been trained on.

In the OSR scenario the classes are typically divided into four sets: Known Known Class (KKC), Known Unknown Class (KUC), Unknown Known Class (UKC), Unknown Unknown Class (UUC). KKC is the set of classes for whom a distinct label is available. In our case, it is the set of labels (the frame IDs) associated with the frames on which the model was trained on. On the contrary, UUC is the set of the classes on which the classifier has never been trained on and for whom no side semantic information is available at training time. KUC and UKC are out of the scope of this work.

Being an OSR, we cannot evaluate our tool with metrics commonly accepted for standard ML close-world problems, such as the accuracy and F1-Score. New accuracy metrics for OSR tasks are firstly presented by Júnior *et al.* [155] and respectively reported in Equation (17) and Equation (18), namely Accuracy for KKC (AKS) and Accuracy for UUC (AUS):

$$AKS = \frac{\sum_{i=1}^{C}(TP_i + TN_i)}{\sum_{i=1}^{C}(TP_i + TN_i + FP_i + FN_i)} \tag{17}$$

$$AUS = \frac{TU}{TU + FU} \tag{18}$$

In Equation (17), TP_i, TN_i, FP_i, FN_i correspond respectively to the number of true positives, true negatives, false positives, and false negatives for the i-th KKC $i \in \{1, ..., C\}$, where C corresponds to the cardinality of KKC. Note that FN_i includes the samples of KKC wrongly classified as unknowns. In Equation (18), true unknowns TU correspond to the samples of UUC correctly classified as unknown, while false unknowns FU correspond to the samples of UUC wrongly classified with one of the KKC labels.

10.3.1 Classifiers

OSR problems are mainly addressed in two ways: (i) by enabling a common ML classifier to reject unknown samples, and (ii) by employing a classifier inherently designed to deal with UUC.

Regarding (i), we adapt a RF [156] classifier and a Fully Connected Neural Network (FCNN) [157] to reject samples whose confidence score for the predicted label is below a certain *rejection threshold*, in the way described in [154]. We choose RF due to its robustness to outliers and the consistent handling of unbalanced datasets, which is particularly relevant in our case. After a tuning on the RF, we discovered that the optimal performance is obtained with around 200 tree estimators. After tuning it by using the implementation provided by the Python library scikit-learn [158], we discovered that no significant performance improvement is obtained with more than 200 tree estimators.

Regarding FCNN, we choose it for its robustness to outliers and the overall superior performance of neural networks compared to traditional ML classifiers documented in literature [157]. After an extensive tuning on the number of layers and neurons for each layer, we found out that having more than three hidden layers with more than 1024 neurons each (and ReLu activation function) does not improve the overall performance of the model. For the implementation, we use the Python library Keras built on Tensorflow [159].

For what concerns (ii), we have reviewed the main inherently-Open Set classifiers in literature. The vast majority of related works specifically addresses computer vision tasks – as OSR is especially relevant for this field – and, therefore, the new classifiers are designed accordingly. In this regard, a particular effort is put into the adaptation or design of new CNN architectures [154], [160]. However, since our data is characterised by a maximum of six features, as opposed to the high-dimensional data typically processed in computer vision tasks, the majority of the methods presented in literature are unsuitable for our task.

For this reason, we selected two algorithms, PI-SVM [161] and Extreme Value Machine

(EVM) [162], which can process input with different spatial properties without the need of data dimensionality transformations. PI-SVM integrates the notions of data distribution from Extreme Value Theory (EVT) [163] into the widely known SVM classifier. EVM is a brand new algorithm, which also exploits the EVT theory about data distribution to group samples in consistent areas of the space.

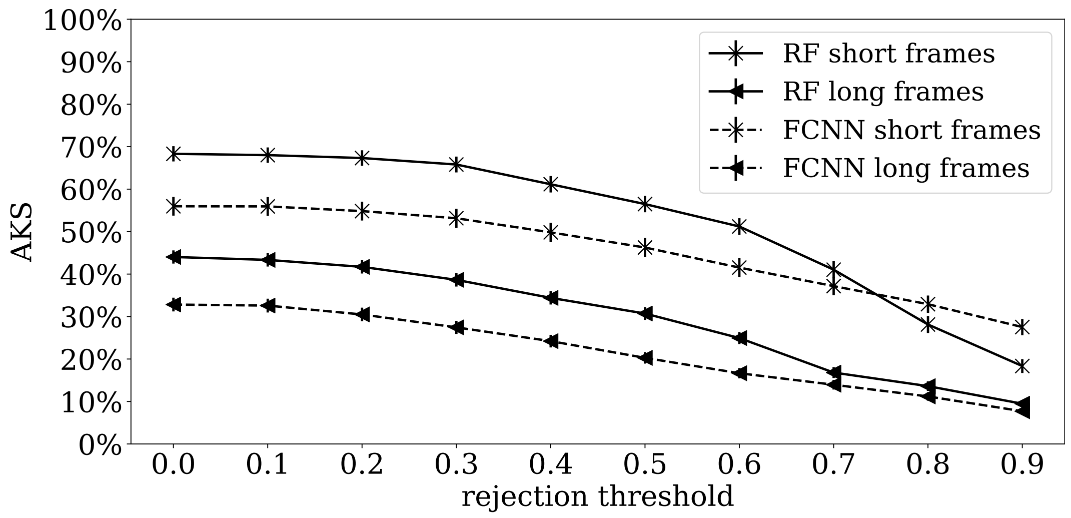
10.3.2 Comparison between classifiers

We first analyse the results obtained by RF, FCNN, EVM, and PI-SVM to assess which classifier provides the best performance overall. After testing PI-SVM and EVM based on the source code released by the authors and the settings suggested in the respective papers [161], [162], we performed an extensive tuning of the hyperparameters to optimise their performance. The tuning revealed that EVM and PI-SVM score AKS and AUS lower than 20 %. Given their poor performance, we further show only the results obtained with RF and FCNN.

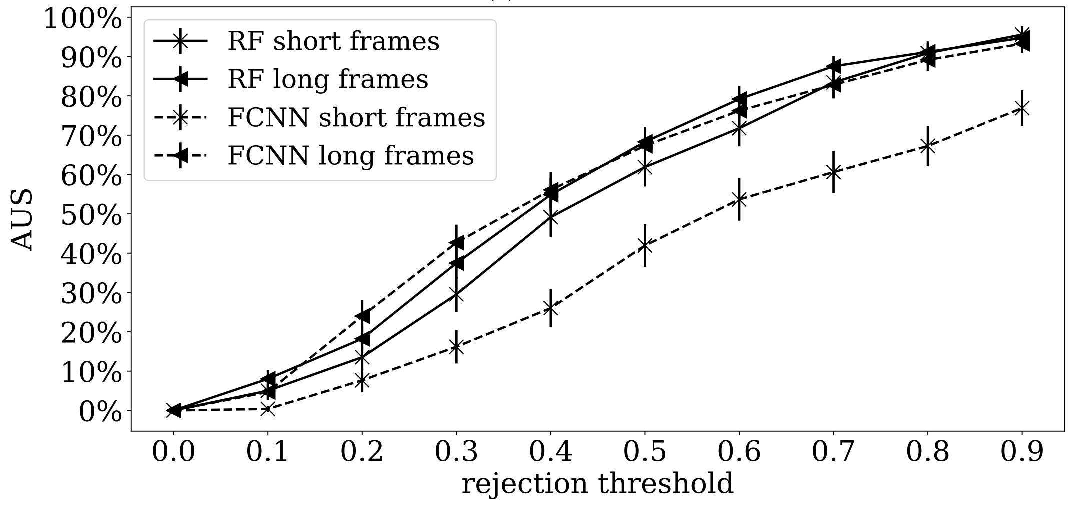
Figure 38 shows the average AKS and AUS, along with 95 % confidence intervals, obtained testing RF and FCNN on the evaluation dataset. The figure highlights that AKS and AUS can vary greatly according to the choice of the rejection threshold. As expected, an increase of the rejection threshold corresponds to a decrease of the AKS and an increase of the AUS. As a matter of fact, the higher the threshold, the more samples are rejected. This causes an increase of the samples in KKC wrongly rejected (hence, the worsening of the AKS), and a decrease of the samples in UUC wrongly labeled as known (hence, the improvement of the AUS).

RF performs equal or better than all the other classifiers on both samples from the short and long frames subsets on all considered metrics and datasets. RF and FCNN achieve a similar AUS for what concerns samples from short frames. It is interesting to notice that, for high values of the rejection threshold, FCNN achieves a higher AKS compared to RF when considering samples from both the short and the long frames.

It has been shown in [164] that the probabilistic output provided by most classifiers is skewed towards the extreme values (0 and 1) and, thus, does not enable an understanding of the true probability of the prediction correctness. It follows that the predictions probabilities outputted



(a) AKS



(b) AUS

Figure 38: Comparison of the performance achieved by RF and FCNN for different rejection thresholds.

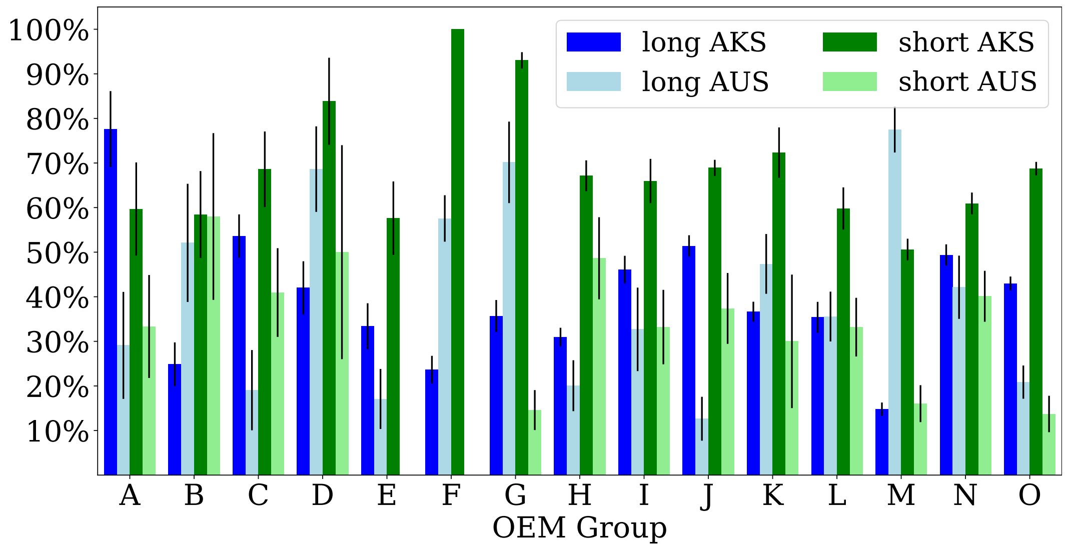


Figure 39: Performance of RF with a rejection threshold of 0.3 achieved on the vehicles of each OEM group.

by the selected classifiers do not correspond to the actual probability of a sample belonging to a determined class. Hence, since different classifiers calculate the probabilistic scores according to different logic, the setting of a certain rejection threshold can have consistently different impact on AKS and AUS.

10.3.3 Result Analysis

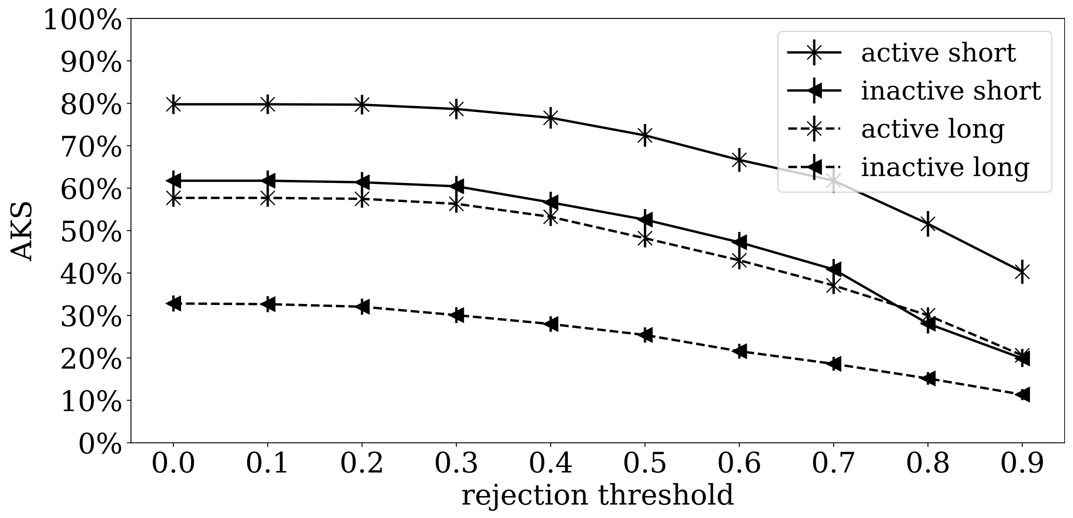
Having designated RF as ultimate classifier for the task, based on the superior classification accuracy compared to the other algorithms, we further investigate its performance. In particular, we analyse the results obtained on the different industrial groups/alliances. Figure 39 illustrates the average AKS and AUS, with 95 % confidence intervals, obtained by RF with a rejection threshold of 0.3 on all vehicles for each industrial group/alliance. The results show a high variance in the average performance scored by the models trained on vehicles from different industrial groups for all considered metrics. We have not found any clear correlation between the classification accuracy and any other characteristic related to the composition of the sample sets for each industrial group reported in Table 15.

Finally, we investigate the importance of the features Percentage of Active Bits and Mean Bit Rate. Given that the testing set is composed of parked vehicles, numerous signals are never triggered and their bits never flip. Such signals are, for instance, all those related to the steering wheel or the vehicle speed. Following an analysis of the dataset, we discovered that, due to this phenomenon, the traces have 53.8 % of frames whose bits in the payload never flip. We refer to these frames as *inactive.* In contrast, the frames in which at least one bit flips during data collection are called *active.* The consequence of having inactive frames is that both the Percentage of Active Bits and Mean Bit Rate of the samples extracted from them are 0, thus making these two features irrelevant for their fingerprinting. By extension, we refer to the samples generated from the active and inactive frames as, respectively, *active* and *inactive samples.*

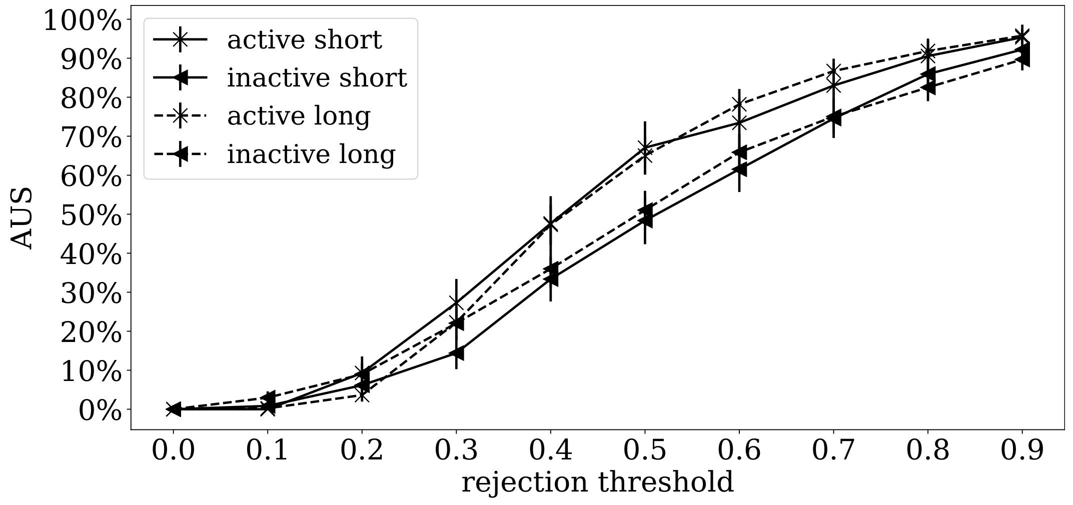
In this scope, we investigate the impact that inactive samples have on the overall classification performance. Figure 40 compares the mean AKS and AUS obtained by the classifier on active and inactive samples according to different rejection threshold. The results show that RF achieves higher AKS and AUS on active samples compared to inactive samples. For instance, with a no-rejection threshold (i.e. equal to 0), the classifier achieves an AKS of 79.8 % on active samples extracted from the short frames set, compared to 61.7 % scored on inactive samples from the same set, thus marking a difference of 18.1 % in the performance. Still assuming no threshold, an even larger gap of almost 25 % remarks the difference in the performance achieved by the classifier on active samples compared to inactive samples extracted from the long frames set.

Interestingly, this gap decreases when the rejection threshold increases in the case of samples extracted from the long frames set, while it slightly augments when considering samples from the short frames set. The conclusion is that the activeness of the bits is more relevant for fingerprinting samples related to short frames compared to long frames.

The presented results show that the features Percentage of Active Bits and Mean Bit Rate impact greatly on the classification performance. This fact suggests that the classification can be sensitively more accurate if the classifier is trained on samples from traces collected on vehicles that are driven.



(a) AKS



(b) AUS

Figure 40: Comparison of the mean AKS and AUS obtained by RF on active and inactive samples.

10.4 Discussion

In this section, we studied whether anonymising the CAN frame IDs makes frame matching based CAN bus reverse engineering methods ineffective, thus preventing vehicle-agnostic remote attacks on CAN bus. We tested the efficacy of this approach by fingerprinting frames whose IDs are anonymised. In this scope, we trained ML classifiers to recognise frames based on six features extracted exploiting DLC, bit flipping, sending frequency and frame priority. We evaluated our fingerprinting method on 33 034 samples extracted from traces collected on 427 different vehicle models.

The results show that an RF classifier can recognise frames with a 8 Byte-long payload and frames with a shorter payload with a mean AKS up to 44.1 % and 68.7 % respectively. When considering samples extracted from frames with at least one flipping bit, a superior AKS up to 57.7 % and 79.8 % is achieved on samples extracted from long and short payload frames respectively. The conclusion is that the fingerprinting can be consistently more accurate if the CAN traces are extracted in a dynamic context (moving vehicles). The presented results highlight that anonymising the CAN frame IDs does not prevent reverse engineering based on frame matching. This suggests that connected vehicles are vulnerable to the remote decoding of CAN data by potential adversaries who can then inject attacks with minimal effort. To preserve better the anonymity of the CAN data transiting in-vehicle networks, substantial changes in the CAN encoding practices is necessary.

Future work includes research for new features able to increase the fingerprinting performance.

11

Preventing Frame Fingerprinting through Traffic Mutations

In Section 10 we presented the scenario in which frame IDs are anonymised for each newly released vehicle model, and we investigated whether this solution is sufficient to effectively prevent frame matching-based CAN bus reverse engineering. In particular, we employed ML classifiers to perform statistical traffic analysis on the CAN bus and fingerprint frames according to the patterns that emerge from their time series. The results showed that it is still possible to recognise anonymised frames with an accuracy up to 80 %.

In this section, we study whether it is possible to make CAN frames less identifiable through traffic mutations, i.e. altering the traffic in such a way that the resultant distribution of values over certain properties of the frames becomes more similar among different frames. The goal of such operations on the CAN traffic is to reduce the recognisability of frames, thus making the ID anonymisation an effective defence against frame matching-based fingerprinting.

11.1 Introduction to Traffic Mutations

The goal of traffic mutation algorithms is to conceal the patterns that emerge from traffic features, which ML models can learn and exploit to recognise sensitive information. The two main techniques adopted in the scope of fingerprinting prevention are *padding* and *morphing.*

Padding consists in augmenting the length of packets in a communication stream to a pre-defined target size. It was introduced by Liberatore and Levine [165] in response to websites fingerprinting attacks over encrypted traffic based on the size of transmitted packets. The goal of morphing, instead, is to make a set of source processes that need to be protected from fingerprinting resemble another target process, hindering the predictive capability of ML classifiers. In its original implementation [166], morphing matrices are generated offline with convex optimisation methods to define how the packets of the source processes should be padded or split.

Traffic mutation approaches have been used as a defence in a multitude of fingerprinting attack scenarios, such as web encrypted traffic [165]–[169], mobile devices and apps [170], VoIP data [33], and Internet of Things (IoT) devices [171], [172]. Among those, it is worth mentioning the adoption by The onion router (Tor) of traffic padding between the client and network entry guard as a countermeasure against website fingerprinting [167].

The specifications of the CAN protocol, as well as the constraints that it is subject to, make the data transiting within vehicles differ consistently from the traffic on which mutations techniques have been designed in literature. Namely, the CAN traffic is not encrypted and frames series bring sequential information. In addition, the length and the sending frequency of the frames in a frame series are pre-defined and do not change over time. In this study, we present novel padding and morphing techniques that fit the unique characteristics of the CAN bus.

11.2 Methodology

The frame fingerprinting approach presented in Section 10 revolves around four distinctive aspects of the CAN traffic: the payload length, the dynamic behavior of the payload bits, the sending frequency, and the frame priority. In that case, the following assumption regarding the frame priority was made: the attribution of the new set of frame IDs in newly released vehicle models should preserve their priority. This leaves no space for manipulation against frame fingerprinting. Therefore, in this work, we operate on the three remaining aspects: (i) padding of the frame payload length, (ii) morphing of the frame sending frequency, and (iii) morphing of the dynamic behavior of the payload bits.

11.2.1 Payload Length Padding

The frame length, expressed by the DLC field, is a feature that can be used to fingerprint frames whose payload is shorter than 8 Byte – the maximum frame length. Hereafter, we refer to this set of frames as *short* and to those whose length is 8 Byte as *long*. In this work, we make the following assumptions:

- The length of the payload can be only increased, as a reduction would cause a loss of information.
- The extra padding bits should be appended at the end of the frame to preserve the original location of the signals within the payload. This allows the receiving ECU to correctly interpret the actual signals contained in the payload and discard the rest of the bits.

Based on the assumptions, we study two solutions:

(a) All payloads are padded to the maximum length of 8 Byte. This is the most straightforward solution, but which also adds the most overhead.

(b) Short payloads, whose length is inferior to a certain threshold τ are padded to τ, while those whose length is superior than τ are padded to 8 Byte length, where $\tau < 8$ Byte. The parameter τ should be chosen to be an optimal trade-off between reduction of fingerprinting accuracy and added overhead.

Since each vehicle has a different distribution of long and short frames – as well as different distribution of payload length among the short frames – we argue that an optimal universal threshold τ cannot be identified. On the contrary, it is reasonable to define τ for each vehicle based on the distribution of the frames lengths. For this purpose, the algorithm calculates the quartiles Q of payload lengths of all frame series and assigns τ based on the value of a chosen quartile Q_i.

Algorithm 4 summarises the methodology adopted to pad the frames payload. Given a reference trace R containing an example of the original traffic of the vehicle, the threshold τ is initially calculated based on the DLC of all frames (lines 1-2). Each frame sent by any ECUs is padded to τ if its length is inferior to τ, or to 8-bytes otherwise (lines 3-7).

To designate the content of the padded bits, multiple approaches can be followed:

1. Set all bits to a constant value, i.e. always 0 or 1;

2. Define the status of each bit randomly, for each frame;

3. Adopt a particular heuristic to set the bit values.

We argue that approach (1) would allow to easily identify how many bits have been padded for each frame, thus nullifying the benefits of the padding. As a consequence, in this work we evaluate approaches (2) and (3). In particular, regarding (3), we follow the algorithm presented in Section 11.2.3.

11.2.2 Sending Period Morphing

Frames having the same ID and that carry periodic signals are typically being sent according to a pre-defined period. However, due to hardware imprecision of clocks embedded within ECUs, the frames deviate from the target sending frequency. On the one hand, this deviation (or *offset*) from the defined sending frequency can be used as an intrusion detection mechanism [39]. On the other hand, the frame sending period and the corresponding offset can be exploited as a feature for frame fingerprinting, as demonstrated in Section 10.2.

In this work, we make the following assumptions:

Algorithm 4 Length Padding

```
Input: Ref. Trace $R$, Threshold $\tau$
Output: Length-Padded Frames
1: for $frame$ in $CAN\_traffic$ do
2:     if $frame.DLC < \tau$ then
3:         $padded\_frame \leftarrow \text{pad}(frame,\ \tau)$
4:     else
5:         $padded\_frame \leftarrow \text{pad}(frame,\ 8)$
6:     end if
7: end for
```

1. The sending period of the frames can be only reduced (i.e. the sending frequency is increased). In fact, incrementing the sending period of frames carrying critical information about the vehicle status would cause ECUs reacting less promptly, thus threatening the vehicle's safety.

2. The car manufacturer can synchronise all ECUs according to the new set of sending frequencies of all frames.

Based on these assumptions, we design Algorithm 5 to morph the sending frequency of all frames. The algorithm takes in input a reference CAN log R and an integer P. R is a sample of the original traffic of the vehicle with no mutation applied. P defines the number of target sending periods to which the frames series will be morphed. If we assume P_o is the number of discrete sending periods extracted from the original trace R, then we must have $P < P_o$.

R allows OEMs to preliminarily collect sending periods and calculate the mean standard deviation of the sending periods of all frames (lines 1-2). Then, the set of sending periods of R is ordered and divided in P quantiles. The ECUs are set to send the frames according to the target periods, which are computed based on these quantiles (line 3). In particular, each frame ID must be associated with a target period such that (i) the target period is inferior to its original period (in compliance with assumption (1)), and (ii) the difference between the target period and the original period is minimal in order to reduce the traffic overhead.

But since we know that each ECU introduces a distinct offset that can help fingerprinting its frames, an additional defence mechanism is needed. Specifically, for each target period

Algorithm 5 Send Frequency Morphing

```
Input: Ref. Trace $R$, # of Target Sending Periods $P$
Output: Send Period-Morphed Frames
1: $periods \leftarrow$ get_periods($R$)
2: $offsets \leftarrow$ extract_offsets($R$, $periods$)
3: $target\_periods \leftarrow$ quantiles($periods$, $P$)
4: $target\_offsets \leftarrow$ assign($R.frame\_ids$, $target\_periods$)
5: $target\_K \leftarrow$ compute_K($new\_periods$, $target\_offsets$)
6: for $frame$ in $CAN\_traffic$ do
7:     $morph\_frame \leftarrow$ morph($frame$, $target\_offsets$, $target\_K$)
8: end for
```

the highest standard deviation among all frame series associated to that period is chosen as a target mean offset (line 4). Let σ_c be the standard deviation of a frame series F, π its target period, and σ_t its target standard deviation. To morph F to have overall sending frequency standard deviation equal to σ_t while keeping its sending period $\sim \pi$, the ECU must aim at sending each frame every $\pi \pm r$, where r is randomly generated following a uniform distribution between $[-K, K]$. K is calculated (line 5) as follows:

$$K = \sqrt{3(\sigma_c^2 - \sigma_t^2)} \tag{19}$$

Finally, the ECUs send the frames according to the new target sending period and offset (lines 6-8).

11.2.3 Bit Flip Rate Morphing

The work in Section 10.1 demonstrates that the number of bits that flip at least once during the trace and the mean BFR constitute valuable features to fingerprint the frames. In the current work, we perform morphing on the payload bits to hide the patterns associated to the BFR.

Every bit within a signal is relevant to the interpretation to information contained in the signal. On the contrary, unused bits (see Section 2.2) are bits that do not belong to any signal and never flip. Since we do not want to alter the actual information carried by the payload, the unused bits are the only ones on which we can operate. In this work, we make unused bits flip in such a way that it makes the overall dynamicity of the frames series to resemble to each other. To be noted that, if payload padding is applied too, the added bits are considered unused and, therefore, can be employed for the morphing.

The pseudocode related to the morphing of BFR is presented in Algorithm 6. The algorithm gets in input a reference trace R, the list of signals S, and an integer B. Similarly to the sending frequency morphing (see Section 11.2.2), R is a sample of the original traffic of the vehicle. S is the list of signals that can be found in R, as reported in the specifications owned by the OEM. B defines the number of target mean BFRs to which the frames series will be morphed.

The frames in a series can only be mutated in such a way that the overall mean BFR is higher than the original one. In fact, trying to decrease the BFR would necessarily imply a manipulation of the bits of the signals, thus causing a loss of information. As a consequence, R should be dynamic enough to ensure that the original mean BFR of the frame series in the CAN traffic can be morphed to the B targets.

The mean BFR of all frames series are initially calculated on R (line 1). The set of all mean BFRs of R is ordered and divided in B quantiles. The target mean BFRs are then extracted according to the quantiles (line 2). Subsequently, for each frame series, the unused bits are extracted based on the ground truth (line 3). Finally, the BFR of each frame series in the CAN traffic can be then morphed to its target (lines 5-11).

To morph a frame series F to have a mean BFR equal to a target mean BFR T, a number of unused bits n must flip for each frame in F. Prior to sending a frame F_i, the ECU defines n by taking into consideration the BFR achieved until F_{i-1} and the information encapsulated in F_i (line 6). Once n is calculated, a subset S of the unused bits in the frame is selected, such that $|S| \leq n$ (line 7). All the bits indicated by S are then flipped and the so altered frame is sent (line 8).

11.3 Evaluation

We evaluate the efficacy that each of the techniques presented in Section 11.2 have against frame fingerprinting singularly and combined. To validate our approach, we employ the same testing set used for the evaluation of the fingerprinting approach presented in Section 10.2. Since the presence of inactive frames in our dataset impacts the fingerprinting performance evaluation – namely, the classifier cannot exploit the features based on the payload dynamicity – we present the results achieved on active and inactive frames separately. For the validation, we follow a leave-one-out-cross-validation approach. In addition, for all the presented results we report the confidence interval at 95%.

To assess the fingerprinting performance, we choose AKS as a metric. For the fingerprinting task, we choose a RF classifier [156] with a depth of 200, which was proven to achieve the

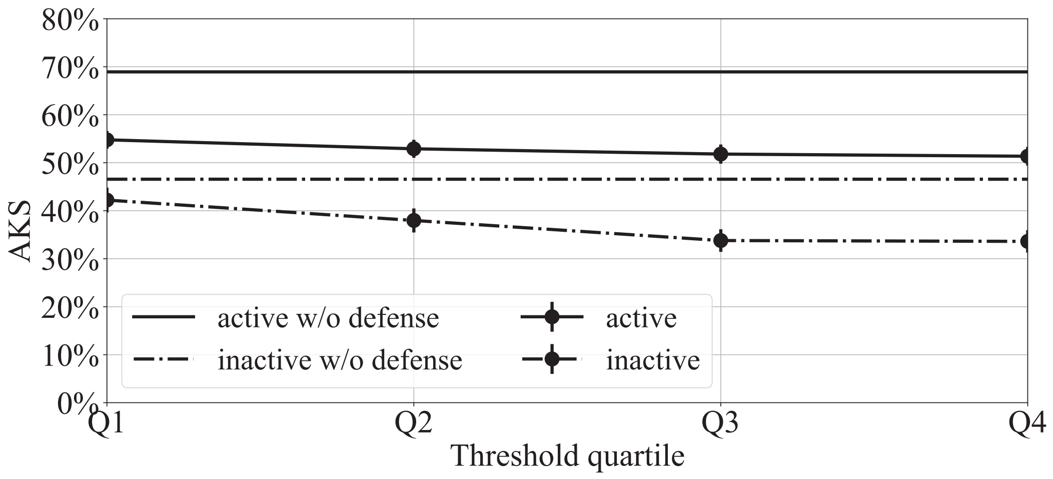


Figure 41: Comparison between the AKS on the original traffic and payload-padded traffic for different quartile values chosen to set τ.

highest AKS in the scope of frame fingerprinting among a set of selected classifiers, such as EVM, PI-SVM, and FCNN Section 10.2. Given the presence of unknown samples, the classifier is set with a rejection threshold = 0.3, i.e. all predictions with a confidence score inferior to 20 % are discarded.

11.3.1 Evaluation of payload length padding

We evaluate the impact that padding the payload of the frames has on the fingerprinting accuracy of the RF classifier. For this analysis, we set the algorithm to choose the content of the padding bits randomly. Figure 41 compares the mean AKS obtained on the original traces (without defence) with the results achieved by applying payload padding on all the tested vehicles (with our proposed defence mechanism in place). The figure shows how the performance varies based on the choice of the quartile used to the determine the threshold τ (see Section 11.2.1). It is to be noted that $Q4$ represents the case in which all frames are padded to 8 Byte (solution (a), Section 11.2.1). The figure highlights that, for both active and inactive frames, the defence improves when the quartile to determine the threshold is higher. It is worth noting that the choice of τ impacts more the fingerprinting on the inactive frames (loss of accuracy of 4 % and 13 % for, respectively, Q_1 and Q_4) than on the active frames (loss

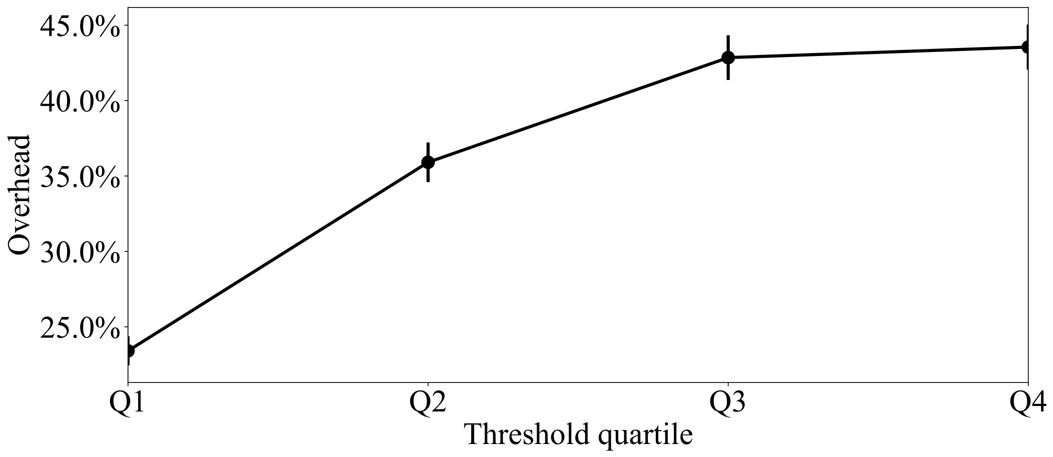


Figure 42: Communication overhead introduced by payload-padded traffic for different quartile values chosen to set τ.

of accuracy between 14 % and 18 %).

We also evaluate the impact that padding the payload of the frames has on the transmission overhead. In particular, Figure 42 shows that the overhead increases from a minimum of 23.4 % (Q1) to a maximum of 43.5 % when all payloads are padded to 8 Byte. This means that slightly improving the defence mechanism comes at a higher cost in terms of communication overhead on the CAN bus.

11.3.2 Evaluation of sending period morphing

We evaluate the efficacy of morphing the frame sending period against frame fingerprinting by comparing the results with the performance obtained on the original non-morphed CAN traffic. Figure 43 illustrates how the fingerprinting performance varies according to the number of quantiles P used to select the target periods (see Section 11.2.2). The figure highlights that the fingerprinting accuracy increases with P. The reason is that the higher the number of target periods, the closer the sending period of each frame is to its original value on average. The fingerprinting performance on the active and inactive frames is reduced by a maximum of, respectively, 9.5 % and 6.5 % for $P = 1$, and it follows a similar trend when varying P. As a matter of fact, this morphing technique does not alter the content of the frames and, therefore, impacts similarly the active and inactive frames.

Algorithm 6 BFR morphing

```
Input: Ref. Trace $R$, Signals $S$, # of Target Mean BFR $B$
Output: BFR-Morphed Frames
 1: $BFRs \leftarrow$ get_mean_bfr($R$)
 2: $target\_BFRs \leftarrow$ quantiles($BFRs$, $B$)
 3: $unused\_bits \leftarrow$ extract_unused_bits($R$, $S$)
 4: for $frame$ in $CAN\_traffic$ do
 5:     $curr\_frame\_series \leftarrow CAN\_traffic[frame.id]$
 6:     $curr\_BFR \leftarrow$ bfr($curr\_frame\_series$)
 7:     $n \leftarrow$ n_bits_to_flip($frame$, $curr\_BFR$, $target\_BFRs$)
 8:     $S \leftarrow$ bits_to_flip($n$, $unused\_bits$)
 9:     $morph\_frame \leftarrow$ morph($frame$, $S$)
10: end for
```

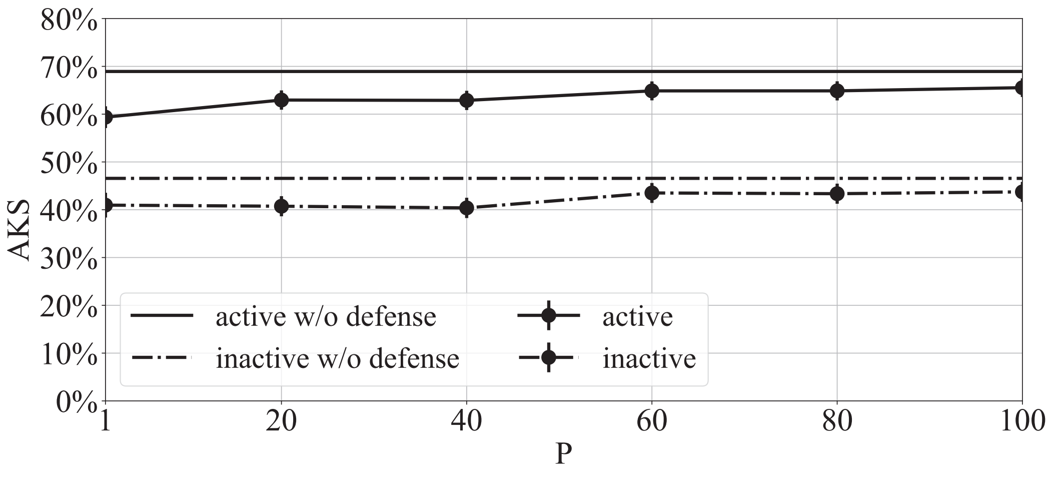


Figure 43: Comparison between the AKS on the original traffic and sending *period-morphed* traffic for different values of P.

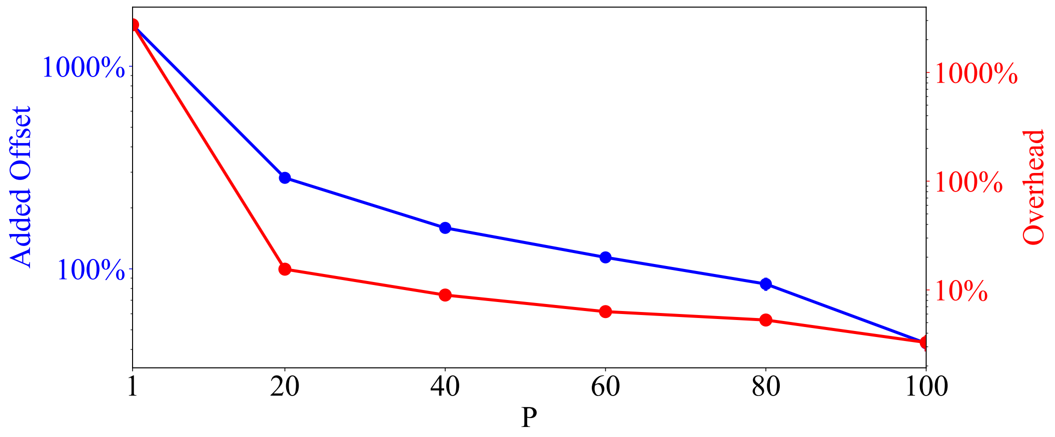


Figure 44: Overhead and added deviation from the target sending period according to the quartile chosen to set τ.

Figure 44 shows how morphing the sending period of frames impacts the performance of the CAN transmission in terms of added overhead and the mean clock offset added. The figure highlights that the number of reference sending periods is inversely proportional to the overhead and to the added deviation. In fact, the more reference periods, the inferior is the difference with the original sending period on average, and less frames are morphed to high offset targets.

11.3.3 Evaluation of BFR morphing

We evaluate the efficacy of BFR morphing against fingerprinting, by comparing the accuracy obtained by the classifier on the morphed CAN traffic with the results achieved on the original traffic at the varying of B. Given the incompleteness of the ground truth, for this evaluation we consider as inactive all frames that do not belong to known signals and that never flip throughout the traces.

Figure 45 highlights that, in the case of active frames, the fingerprinting performance increases with B and, thus, efficacy of the morphing decreases. In particular, for $B = 1$, the accuracy is reduced by circa 11 %, while for $B = 100$, it is decreased by $\approx 3\,\%$. In fact, the higher the number of mean BFR values used as targets for the morphing, the less the difference with the original dynamicity of the frames series. On the contrary, the performance obtained

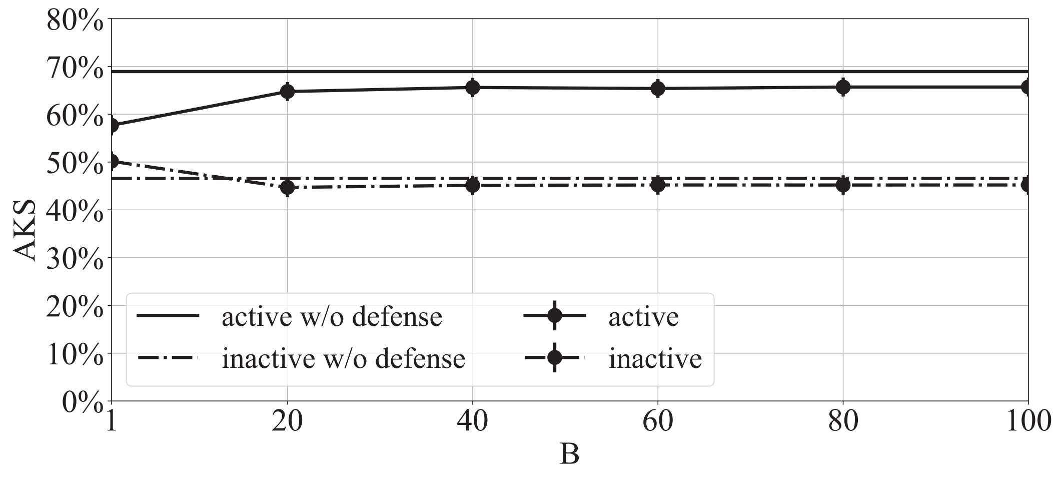


Figure 45: Comparison between the AKS on the original traffic and *BFR-morphed* traffic for different values of B.

by the classifier on inactive frames remains stable. As a matter of fact, the features based on BFR are irrelevant for the fingerprinting of inactive frames. Therefore, activating these frames cannot help in reducing the fingerprinting capability of the classifier. To be noted that, unlike the other two approaches, BFR morphing does not introduce any additional overhead on the communication channel.

11.3.4 Evaluation of combined approaches

After analysing the impact that padding and morphing have on the frame fingerprinting accuracy based on the different characteristics of the CAN traffic, we evaluate the impact that combinations of these approaches have overall. Following the results shown in Sections 11.3.1, 11.3.2 and 11.3.3, we select $\tau = Q1$, $P = 40$, and $B = 1$. We consider this as a fair trade-off between the decrease in the AKS and the negative impact that such operations have on the communication overhead.

Figure 46 compares the fingerprinting accuracy obtained for different combinations of the three proposed approaches with the accuracy achieved on the original traffic. To be noted that when frame padding and BFR morphing are combined, the content of the padded bits is chosen according to the heuristic defined by the morphing. The results presented in the figure

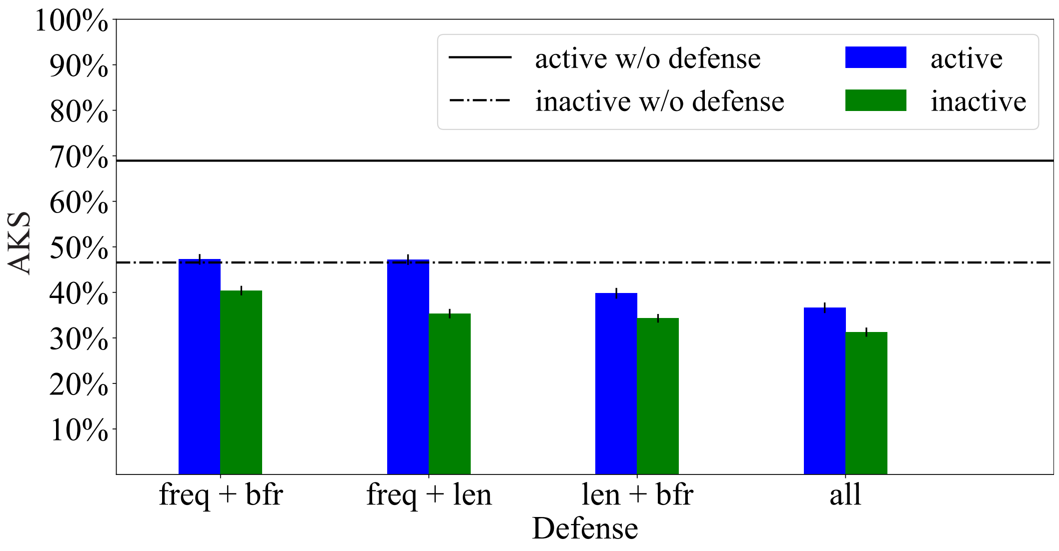


Figure 46: Evaluation of the different combinations of the proposed mutations.

confirm that combining multiple approach leads to lower fingerprinting performance. The best result is obtained when all the operations on the CAN traffic are applied. In particular, the AKS on active and inactive frames shrinks, respectively, from 68.9 % to 36.6 % for active frames and from 46.6 % to 31.2 % for inactive frames.

11.4 Discussion

In this section, we presented a methodology to reduce the efficacy of CAN frames fingerprinting performed by ML classifiers without applying modifications to the protocol. Our approach is based on mutating the CAN traffic by (i) padding the length of frame payloads, (ii) morphing the frame sending frequency, and (iii) morphing the dynamic behavior of the payloads. The proposed methodology is tested and evaluated on real CAN logs from 427 different vehicle models. The performance evaluation shows that our approach reduces the frame fingerprinting accuracy by more than 40 %.

The main limitations of this work concern the overhead added to the transmission – 23 % and 8 % when adopting, respectively, payload padding and sending frequency morphing – and the offset from the target sending period added through sending frequency morphing – above 140 %. Future work includes developing an improved version of the presented methodology to further decrease the frame fingerprinting accuracy, while also reducing the overhead and average frame sending offset.

Part V

CONCLUSION

12 Future Research Perspectives

In this section, we discuss the possible directions that the future research on CAN reverse engineering aimed at improving the performance and optimising the processes, based on our evaluation of the presented techniques. At the time of writing, research on reverse engineering has solely focused on the classic version of CAN 2.0, with no multiplexing and no flexible data rate. As it follows, we propose how to adapt the state-of-the-art approaches to tackle multiplexing and CAN FD. Finally, we address future work on the prevention of CAN reverse engineering.

12.1 Improving CAN reverse engineering

In this dissertation, we analysed the state-of-the-art techniques for semi and fully automated reverse engineering proposed in the literature, as well as novel approaches. As discussed in Section 8, the presented methodologies largely vary in terms of performance, number of properties of the signals decoded, requirements, as well as computational complexity and time needed for the data collection. In the following, we suggest three possible areas of improvement that should be investigated in future research.

12.1.1 Generalisation

Work on CAN reverse engineering has produced promising results in real world-scenarios. For what concerns the translation, recall and precision superior than 80 % were reached by multiple algorithms on different vehicles ([23], [24], [29], Sections 5 and 7). However, oftentimes the performance consistently varies across different vehicles. CAN reverse engineering should be reliable and consistent when considering any vehicle model.

Furthermore, little research has been conducted on EVs, and none on motorbikes, hydrogen-propelled vehicles, and autonomous cars. The types of ECUs embedded in these categories of vehicles can differ consistently from those on which the literature has focused. For instance, the ECUs related to the powertrain functions in EVs and hydrogen vehicles bring intrinsically different types of information compared to gas-propelled cars. As a consequence, supervised ML and taxonomy-based approaches need to be adapted to take into account this diversity. Moreover, since PIDs were born with the intent of monitoring emissions in oil and diesel vehicles their usage is limited on EVs and hydrogen vehicles.

A comparative analysis should be also conducted to understand how the performance vary according to the number of ECUs attached to the bus. Intuitively, to a low number of ECUs correspond fewer types of frames transiting on the bus and, thus, a reduction in the complexity of the data to analyse and a decrease in the translation mislabelling. Given that motorbikes have typically fewer ECUs compared to cars and trucks [173], we can expect higher reverse engineering performance on them. Vice versa, the reverse engineering would likely be more difficult on autonomous vehicles due to the higher presence of ECUs [174], which are necessary to handle the complexity of the autonomous drive. The validation should be done taking into account the market segment of the vehicle. In fact, high-end vehicles transmit a greater variety of CAN frames compared to low-end ones, due to the higher number of ECUs installed [175].

To improve the robustness and the performance of reverse engineering, we recommend a combination of the approaches discussed in this dissertation. PIDs could be used to preliminarily decode all signals for which ground truth is publicly available. A taxonomy-based approach could be then employed to reverse engineer signals missed in the first step. Finally, unsupervised and/or supervised ML problems could identify the signals sharing redundant information with those already decoded. Using this combined approach, a diverse dataset of DBC files would be generated. Frame matching could be employed on new target vehicles to efficiently reverse engineer frames already known. The previously used techniques could still

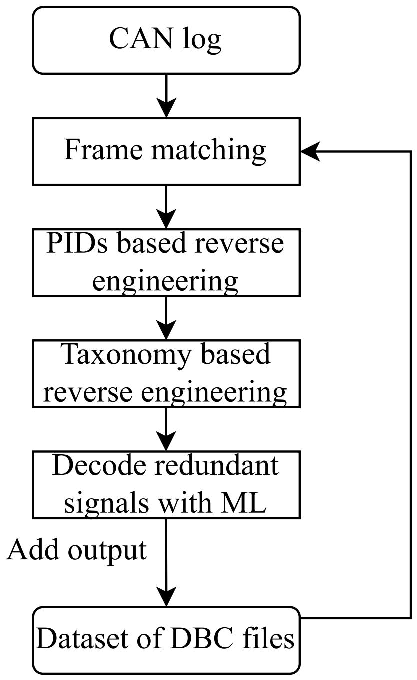

CAN log
Frame matching
PIDs based reverse engineering
Taxonomy based reverse engineering
Decode redundant signals with ML
Add output
Dataset of DBC files

Figure 47: Proposed pipeline for combined reverse engineering.

be employed to augment the DBC dataset and, thus, improve the performance of the frame matching, as illustrated in Figure 47.

12.1.2 Modularity

Every year new ECUs are introduced on the market to offer the latest automotive features and services, from infotainment to connectivity. These ECUs send frames containing novel signals carrying new information semantically different from what has been previously encountered. Reverse engineering tools should be designed following a modular architecture to allow the continuous release and integration of updated software to address these new frames. As an example, in supervised ML-based tools the models should address OSR, i.e., being capable of discarding samples from unknown classes, or being iteratively re-trained/updated to include new classes of signals, and be online, i.e., being able to update incrementally. For what concerns taxonomy-based approach, this upgrade would involve not solely the software, but also the integration of additional steps in the pipeline for the data collection.

12.1.3 Remote reverse engineering

Until recently, CAN reverse engineering has been performed by having physical access to the vehicle. While most of the works analysed in this dissertation allow decoding the CAN traffic completely remotely once the log has been extracted, data collection has always been performed with the presence of a human operator inside the vehicle. Today, there are several dongles that allow remote collection of CAN data [176]. These dongles usually connect to the OBD-II port and send the data through a wireless interface to the cloud for processing and storage.

We argue that collecting CAN data remotely can improve the performance of reverse engineering, since it allows collecting traces in a variety of driving scenarios over time. This is especially true for taxonomy-based methodologies that can potentially become fully automated. Instead of having a human operator wittingly performing the actions needed to collect the data, it would be possible to identify scenarios where the same actions are carried on by logging the sessions of an unaware driver over a long period. A context-aware model, i.e., a representation of typical driving scenarios, could help isolate subsets of signals and trigger the related function able to reverse engineering them. Following the suggestion of combining multiple reverse engineering approaches (see Section 12.1.1), this context-aware model could be a supervised ML model. For example, it would be possible to isolate and decode the signals related to the wipers by comparing logs collected in sunny days with others in rainy days with the help of weather data.

12.2 Reverse engineering vs Multiplexing

Multiplexing in CAN can consistently impact the performance of existing methodologies, in particular tokenisation algorithms. In fact, the assumption on which tokenisation approaches are based is that consecutive frames identified by the same ID carry the same signals within the same positions in the payload. When multiplexing is implemented, this assumption is no longer valid. All algorithms presented in Section 3.3 output a single set of tokens uniquely identified by their start and end position. Since different sets of tokens can be associated with

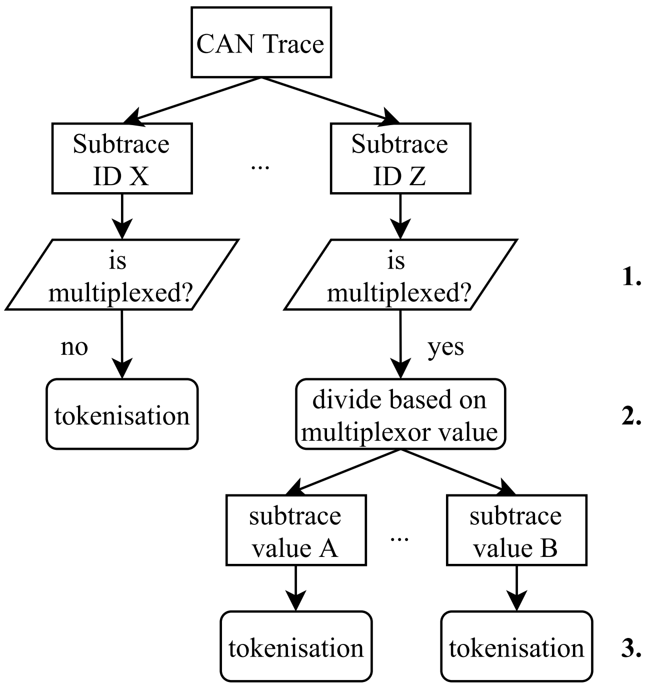


Figure 48: Proposed pipeline for reverse engineering CAN with multiplex frames.

the same ID in the case of multiplexing, such methodologies would produce spurious results. For instance, in case of BFR-based algorithms, the bits of the payload carrying different signals flip independently from one another, thus leading to inconsistencies in the BFR array.

However, we argue that, by following the steps presented in Figure 48, current methodologies can be efficiently adapted to deal with multiplexing. Instead of evaluating the time series of an ID as a whole, researchers could focus on preliminarily identifying whether the payload is multiplexed (step 1) and, if yes, locate the multiplexor signal(s). Step 1 could be achieved, for instance, by observing the sending period of the frames in the time series and assess whether the frames are only sent according to a specific period or they also include a delay with respect to the base cycle. Another technique could be looking for correlations between the flipping of certain bits of the payload, suspected of being the multiplexor and the dynamicity of the rest of the payload. Once the multiplexor is known, it would be sufficient to split the time series into sub-time series, each corresponding to one value of the multiplexor (or combination of values in the case of multiple multiplexors) (step 2), and apply the target tokenisation algorithm (step 3).

12.3 CAN FD reverse engineering

As discussed in Section 2.1.2, the major differences between CAN FD and CAN 2.0 are the superior data transfer rate and the presence of extended frames, whose payload length increases from a maximum of 8 Byte to 64 Byte. A higher data rate does not affect the reverse engineering process as long as the chosen dongle for the data collection is capable of recording all the transiting CAN traffic without loss. For what concerns the extended frames, all the techniques presented in this dissertation can be potentially adapted to take into account the increased length of the payload. However, the resources and time required for algorithms that have a high computational complexity may grow exponentially. As a consequence, we suggest that future research on CAN FD reverse engineering should be conducted and benchmarked especially taking into account the computational complexity.

12.4 Preventing Reverse Engineering

As described in Section 9.2, securing the CAN bus is an extensive topic currently tackled by the scientific community. However, at the time of writing, none of the solutions proposed in the literature has been effectively adopted on large scale to the CAN bus, due to the physical constraints of the network and/or production costs. The increasing connectivity of the vehicles and the ease in decoding CAN data thanks to the recent advancements in the reverse engineering field represent a real threat for the safety of the vehicles and the passengers.

Aside from designing solutions based on cryptography and anomaly detection, researchers should explore specific countermeasures against the reverse engineering, such as the one presented in Section 11. Mutating CAN traffic seems a promising direction to prevent frame fingerprinting and, thus, frame matching. However, our work highlight that traffic mutations add significant overhead to the communication which, in the case of CAN, is critical.

Future work should focus on reducing the overhead while improving the performance of techniques for the prevention of frame matching based on traffic mutations. In addition, traffic mutations and, in particular those based on flipping the unused bits, could be employed to prevent other kinds of reverse engineering. In fact, flipping such bits could help hinder the tokenisation algorithms in identifying the exact boundaries of the signals. Finally, all the new findings should be also tested on CAN FD and multiplexed frames.

13

Final Discussion

CAN bus reverse engineering has emerged as a prominent research topic in the field of in-vehicle communication. Its purpose is to infer the location, semantic meaning, and format of signals transiting on the bus. This output can be employed for real-time interpretation of CAN data, a valuable source of information for researchers and companies developing innovative solutions in the automotive sector. This process has been achieved mostly manually [177]. While having been proven to provide reliable results, the manual approach requires from hours to days of intense human work and constant physical access to the vehicle.

Recently, researchers have started investigating the automation of this process to make it faster, scalable and standardised [23], [27], [28], [30]–[32], [54], [107]. These works either provide a complete pipeline [23] or address one step of the reverse engineering process, i.e. the tokenisation [27], [28], [107] – identification of the boundaries of the signals – or the translation [31], [32] – decoding of the semantic meaning and format of the signals. Related work has demonstrated, with different degrees of success, that it is possible to achieve partial of full automation of the reverse engineering process, thus cutting the time and manual effort for data collection and execution. However, the solutions proposed in the literature work under limited assumptions, address only one aspect of the process, imply a non-negligible human effort and expertise or require a consistent amount of external hardware.

The first goal of this dissertation was to highlight the opportunities and challenges of automating CAN reverse engineering, and explore new solutions to further speed up this process while reducing the requirements in terms of hardware and human effort. In this scope, we initially conducted a comprehensive review of the state-of-the-art methodologies for CAN bus reverse engineering. In particular, this work provides the first categorisation of tokenisation and translation algorithms based on the common ground characteristics of the followed approaches. Additionally, we analysed the strengths and weaknesses of each of these approaches with the intent of providing a benchmark for future research.

Subsequently, we presented three novel reverse engineering algorithms. The first is a complete pipeline for semi-automated reverse engineering based on the taxonomy of the CAN signals. Multiple short CAN logs are collected to isolate groups of semantically related signals, whose decoding is then performed taking into account their specific characteristics. This technique permits a simplification of the data collection and a reduction of the time required for it from the current standard of up to an hour to few minutes. 80 % of 24 distinct signals were correctly decoded based on a set of real CAN logs collected from 5 vehicles.

The second, CSI is a supervised ML-based translation algorithm whose goal is to decode the semantic meaning of a subset of CAN signals considered critical – the foreground set –, among a larger pool of signals – the background set. Based on logs from 8 distinct vehicles, CSI was evaluated using a RF classifier to identify a set of 10 foreground signals among a set of up to 59 background signals, achieving a mean accuracy up to 93%. However, when tested on a background set composed by all tokens extracted by a state-of-the-art tokenisation algorithm, CSI was found to perform poorly. These results show that the features composing the samples do not sufficiently fingerprint the signals in a real world scenario. Future research is needed to extend the set of features considered by CSI and better identify the critical signals.

The third, CANMatch, is a complete pipeline for fully automated reverse engineering, which exploits the reuse of frames IDs across different vehicle models. CANMatch is composed of three phases: frame matching, tokenisation and translation of redundant signals. The frame matching gets in input ground truth related to a large number of vehicles models, which is then used to match and decode frames in the target vehicle. The tokenisation is based on BFR and it is an improved version of the READ algorithm capable of decoding the endianness of the signals. Finally, the identification of redundant signals is achieved by calculating correlations between signals decoded via frame matching and the tokens. In this phase, clustering is employed to reduce the computational time needed for the correlations.

Based on a set of CAN traces collected on 15 driven vehicles, CANMatch achieved a superior tokenisation performance compared to state-of-the-art LibreCAN and READ. For what concerns the translation, on 477 CAN traces extracted from parked vehicles, our method obtained a mean decoding accuracy of up to 83%. CANMatch achieves comparable performance than state-of-the-art tools while requiring no hardware other than the CAN dongle and reducing the data collection time of one order of magnitude. The main limitation of this approach is the intensive use of CAN ground data (in the form of DBC files) from a number of vehicles which, supposedly, should be obtained through manual reverse engineering. In this scope, recommend a combined usage of CANMatch and less automated reverse engineering tool for the collection and update of the ground truth dataset.

Future research should aim at defining the theoretical limits of the automation of CAN bus reverse engineering in terms of equipment required, time complexity, and output accuracy. Furthermore, researchers and companies should work on improving the current solutions to make them more reliable, fast, and generalised. We also think that, when it is applicable, a combination of multiple approaches could provide better performance than the single ones. Future work should also focus on implementing and assessing the performance of reverse engineering on multiplexed data and CAN FD.

The second goal of this work was to study the implications that the automation of the reverse engineering has on the security of CAN. Our analysis suggests that the full automation of the reverse engineering process impacts negatively the security of the network. In particular, the possibility enabled by frame-matching of performing complete reverse engineering without prior knowledge of the vehicle model nor the driving context can potentially lead to large scale remote attacks. In this scope, we summarised the main attacks on CAN and we evaluated the related countermeasures proposed in the literature to assess whether they are sufficient to hinder reverse engineering as well. The defences proposed in the literature result insufficient and/or commercially-wise unsustainable.

Following the outcome of this discussion, we investigated whether anonymising the IDs of the frames would be sufficient to prevent frame matching-based reverse engineering without applying modifications to the CAN protocol. To prove the efficacy of this simple countermeasure, we conducted a study about the possibility of fingerprinting frames independently from their ID. Six features capturing the intrinsic characteristics of the frames were extracted and exploited by a ML classifier to fingerprint the frames. An evaluation based on real-world OSR assumptions was carried on logs related to 427 vehicle models. It was found that following this approach it is still possible to recognise the frames, and subsequently, apply frame-matching based reverse engineering with an accuracy up to 80%. However, the tests highlight varying degrees of success for different industrial groups. Future work is needed to identify the main factors that influence the performance and the limitations to scalability.

Having demonstrated that the anonymisation of the IDs cannot halt frame-matching based reverse engineering, we studied a solution that does not imply modifications to the CAN protocol. We came out with a novel approach based on mutating the CAN traffic by (i) padding the length of frame payloads, (ii) morphing the frame sending frequency, and (iii) morphing the dynamic behavior of the payloads. The proposed methodology was tested and evaluated on real CAN logs from 427 different vehicle models. The performance evaluation shows that our approach reduces the frame fingerprinting accuracy by more than 40 %. The main limitations of this work concern the overhead added to the transmission – 23 % and 8 % when adopting, respectively, payload padding and sending frequency morphing – and the offset from the target sending period added through sending frequency morphing – above 140 %. Future work includes developing an improved version of the presented methodology to further decrease the frame fingerprinting accuracy, while also reducing the overhead and average frame sending offset.

In conclusion, the security concerns highlighted in this work should be urgently addressed by the scientific community and the OEMs. Specific solutions to prevent the full automation of the reverse engineering process and the novel attacks which are enabled by it need to be designed while taking into account the constraints imposed by the network and the necessities of the manufacturers.

BIBLIOGRAPHY

[1] A. Buscemi, I. Turcanu, G. Castignani, R. Crunelle, and T. Engel, “Poster: A Methodology for Semi-Automated CAN Bus Reverse Engineering,” in *13th IEEE Vehicular Networking Conference (VNC 2021)*, to appear, IEEE, Nov. 2021.

[2] A. Buscemi, G. Castignani, T. Engel, and I. Turcanu, “A Data-Driven Minimal Approach for CAN Bus Reverse Engineering,” in *3rd IEEE Connected and Automated Vehicles Symposium (CAVS)*, Victoria, Canada: IEEE, Oct. 2020.

[3] A. Buscemi, I. Turcanu, G. Castignani, R. Crunelle, and E. Thomas, “CANMatch: A Fully Automated Tool for CAN Bus Reverse Engineering based on Frame Matching,” *IEEE Transactions on Vehicular Technology*, 2021, to appear.

[4] A. Buscemi, I. Turcanu, G. Castignani, and T. Engel, “On Frame Fingerprinting and Controller Area Networks Security in Connected Vehicles,” in *IEEE Consumer Communications & Networking Conference (CCNC)*, Virtual Conference: IEEE, Jan. 2022.

[5] A. Athanasopoulou, H. Bouwman, F. Nikayin, and M. de Reuver, “The disruptive impact of digitalization on the automotive ecosystem: A research agenda on business models, platforms and consumer issues.,” in *Bled econference*, 2016, p. 4.

[6] C. Llopis-Albert, F. Rubio, and F. Valero, “Impact of digital transformation on the automotive industry,” *Technological forecasting and social change*, vol. 162, p. 120 343, 2021.

[7] T. Derenda, M. Zanne, M. Zöldy, and A. Torok, “Automatization in road transport: A review,” *Production Engineering Archives*, vol. 20, pp. 3–7, Sep. 2018.

[8] M. Bertoncello, G. Camplone, P. Gao, *et al.*, “Monetizing car data—new service business opportunities to create new customer benefits,” *McKinsey & Company*, 2016.

[9] T. Toledo, O. Musicant, and T. Lotan, “In-vehicle data recorders for monitoring and feedback on drivers’ behavior,” *Transportation Research Part C: Emerging Technologies*, vol. 16, no. 3, pp. 320–331, 2008.

[10] S. Jafarnejad, “Machine Learning-based Methods for Driver Identification and Behavior Assessment: Applications for CAN and Floating Car Data,” Ph.D. dissertation, University of Luxembourg, Esch-sur-Alzette, Luxembourg, 2020.

[11] xee. (), [Online]. Available: `https://www.xee.com/en/industries/#fleet-management` (visited on 02/04/2021).

[12] U. Fugiglando, E. Massaro, P. Santi, *et al.*, “Driving behavior analysis through CAN bus data in an uncontrolled environment,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 20, no. 2, pp. 737–748, 2018.

[13] L. Nkenyereye and J.-W. Jang, “Integration of big data for querying CAN bus data from connected car,” in *9th International Conference on Ubiquitous and Future Networks (ICUFN)*, 2017, pp. 946–950.

[14] D. K. Nilsson, P. H. Phung, and U. E. Larson, “Vehicle ecu classification based on safety-security characteristics,” in *IET Road Transport Information and Control-RTIC 2008 and ITS United Kingdom Members’ Conference*, IET, 2008, pp. 1–7.

[15] A. Neumann, M. Mytych, D. Wesemann, L. Wisniewski, and J. Jasperneite, “Approaches for in-vehicle communication – an analysis and outlook,” Jun. 2017.

[16] Robert Bosch GmbH. (), [Online]. Available: `https://www.boschautoparts.com/` (visited on 01/13/2021).

[17] “Road vehicles — Controller area network (CAN) — Part 1: Data link layer and physical signalling,” International Organization for Standardization, Standard, Dec. 2015.

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

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

[20] National Instruments Corp. “Controller Area Network (CAN) Overview.” (Sep. 2020), [Online]. Available: `https://www.ni.com/en-us/innovations/white-papers/06/controller-area-network--can--overview.html` (visited on 12/06/2021).

[21] SAE, “Recommended practice for a serial control and communications vehicle network,” *SAE J1939 Standards Collection*, 2010.

[22] 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)*, Austin, TX: USENIX Association, Aug. 2016. [Online]. Available: `https://www.usenix.org/conference/woot16/workshop-program/presentation/burakova`.

[23] M. D. Pesé, T. Stacer, C. A. Campos, E. Newberry, D. Chen, and K. G. Shin, “Librecan: Automated can message translator,” in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, ACM, 2019, pp. 2283–2300.

[24] M. E. Verma, R. A. Bridges, J. J. Sosnowski, S. C. Hollifield, and M. D. Iannacone, “Can-d: A modular four-step pipeline for comprehensively decoding controller area network data,” *IEEE Transactions on Vehicular Technology*, vol. 70, no. 10, pp. 9685–9700, 2021.

[25] H. Wen, Q. Zhao, Q. A. Chen, and Z. Lin, “Automated cross-platform reverse engineering of can bus commands from mobile apps,” in *Proceedings 2020 Network and Distributed System Security Symposium (NDSS’20)*, 2020.

[26] Y. L. Le Yu, P. Jing, X. Luo, *et al.*, “Towards automatically reverse engineering vehicle diagnostic protocols,” in *Proc. USENIX Security*, vol. 2022.

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

[28] M. Marchetti and D. Stabili, “Read: Reverse engineering of automotive data frames,” *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 4, pp. 1083–1097, 2018.

[29] W. Choi, S. Lee, K. Joo, H. J. Jo, and D. H. Lee, “An enhanced method for reverse engineering can data payload,” *IEEE Transactions on Vehicular Technology*, vol. 70, no. 4, pp. 3371–3381, 2021.

[30] M. Verma, R. Bridges, and S. Hollifield, “Actt: Automotive can tokenization and translation,” in *International Conference on Computational Science and Computational Intelligence (CSCI)*, IEEE, 2018, pp. 278–283.

[31] T. Huybrechts, Y. Vanommeslaeghe, D. Blontrock, G. Van Barel, and P. Hellinckx, “Automatic reverse engineering of CAN bus data using machine learning techniques,” in *International Conference on P2P, Parallel, Grid, Cloud and Internet Computing*, Springer, 2017, pp. 751–761.

[32] U. Ezeobi, H. Olufowobi, C. Young, J. Zambreno, and G. Bloom, “Reverse engineering controller area network messages using unsupervised machine learning,” *IEEE Consumer Electronics Magazine*, 2020.

[33] W. B. Moore, H. Tan, M. Sherr, and M. A. Maloof, “Multi-class traffic morphing for encrypted voip communication,” in *International Conference on Financial Cryptography and Data Security*, Springer, 2015, pp. 65–85.

[34] G. Brindescu. “DARPA Hacked a Chevy Impala Through Its OnStar System.” (2015), [Online]. Available: `https://www.autoevolution.com/news/darpa-hacked-a-chevy-impala-through-its-onstar-system-video-92194.html` (visited on 04/02/2021).

[35] S. Jafarnejad, L. Codeca, W. Bronzi, R. Frank, and T. Engel, “A car hacking experiment: When connectivity meets vulnerability,” in *2015 IEEE Globecom Workshops (GC Wkshps)*, IEEE, 2015, pp. 1–6.

[36] C. Miller and C. Valasek, “Remote exploitation of an unaltered passenger vehicle,” *Black Hat USA*, vol. 2015, no. S 91, 2015.

[37] T. Hoppe, S. Kiltz, and J. Dittmann, “Security threats to automotive CAN networks—Practical examples and selected short-term countermeasures,” *Reliability Engineering & System Safety*, vol. 96, no. 1, pp. 11–25, 2011.

[38] W. Wu, R. Li, G. Xie, *et al.*, “A survey of intrusion detection for in-vehicle networks,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 21, no. 3, pp. 919–933, 2019.

[39] K.-T. Cho and K. G. Shin, “Fingerprinting electronic control units for vehicle intrusion detection,” in *25th USENIX Security Symposium (USENIX Security 16)*, Aug. 2016, pp. 911–927.

[40] 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, 2016.

[41] W. Choi, K. Joo, H. J. Jo, M. C. Park, and D. H. Lee, "Voltageids: Low-level communication characteristics for automotive intrusion detection system," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 8, pp. 2114–2129, 2018.

[42] R. G. Engoulou, M. Bellaiche, S. Pierre, and A. Quintero, "Vanet security surveys," *Computer Communications*, vol. 44, pp. 1–13, 2014.

[43] H. Hasrouny, A. E. Samhat, C. Bassil, and A. Laouiti, "Vanet security challenges and solutions: A survey," *Vehicular Communications*, vol. 7, pp. 7–20, 2017.

[44] H. Hartenstein and K. Laberteaux, *VANET: vehicular applications and inter-networking technologies.* John Wiley & Sons, 2009, vol. 1.

[45] D. J. Fagnant and K. Kockelman, "Preparing a nation for autonomous vehicles: Opportunities, barriers and policy recommendations," *Transportation Research Part A: Policy and Practice*, vol. 77, pp. 167–181, 2015.

[46] A. Faisal, M. Kamruzzaman, T. Yigitcanlar, and G. Currie, "Understanding autonomous vehicles," *Journal of transport and land use*, vol. 12, no. 1, pp. 45–72, 2019.

[47] J. Cui, L. S. Liew, G. Sabaliauskaite, and F. Zhou, "A review on safety failures, security attacks, and available countermeasures for autonomous vehicles," *Ad Hoc Networks*, vol. 90, p. 101 823, 2019.

[48] PEAK System. (2020), [Online]. Available: `https://www.peak-system.com/PCAN-USB-FD.365.0.html` (visited on 06/03/2020).

[49] CSS Electronics. (), [Online]. Available: `https://www.csselectronics.com/screen/product/can-bus-logger-canlogger2000/` (visited on 01/13/2021).

[50] SK Pang Electronics Ltd. (2016), [Online]. Available: `https://cdn.hackaday.io/files/257881103313792/PICAN2DUOUGB.pdf` (visited on 11/05/2021).

[51] Xee. (2020), [Online]. Available: `https://www.xee.com/en/` (visited on 05/27/2020).

[52] C. AI. "OpenDBC." (), [Online]. Available: `https://github.com/commaai/opendbc` (visited on 08/05/2021).

[53] D. Frassinelli, S. Park, and S. Nürnberger, "I know where you parked last summer : Automated reverse engineering and privacy analysis of modern cars," in *2020 IEEE Symposium on Security and Privacy (SP)*, 2020, pp. 1401–1415.

[54] M. Jaynes, R. Dantu, R. Varriale, and N. Evans, "Automating ECU identification for vehicle security," in *15th International Conference on Machine Learning and Applications (ICMLA)*, IEEE, 2016, pp. 632–635.

[55] "Information processing systems - Open Systems Interconnection - Basic Reference Model," International Organization for Standardization, Standard, Nov. 1989.

[56] International Organization for Standardization, "Road vehicles — Controller area network (CAN) — Part 1: Data link layer and physical signalling," ISO 11898-1, Dec. 2015.

[57] "Road vehicles — local interconnect network (lin) — part 1: General information and use case definition," Standard, Aug. 2016.

[58] T.-S. Ho and K.-C. Chen, "Performance analysis of ieee 802.11 csma/ca medium access control protocol," in *Proceedings of PIMRC '96 - 7th International Symposium on Personal, Indoor, and Mobile Communications*, vol. 2, 1996, 407–411 vol.2.

[59] "Road vehicles — FlexRay communications system — Part 1: General information and use case definition," International Organization for Standardization, Standard, Feb. 2013.

[60] G. Miao, J. Zander, K. W. Sung, and S. Ben Slimane, *Fundamentals of Mobile Data Networks*. Cambridge University Press, 2016.

[61] I. Chlamtac and A. Ganz, "Channel allocation protocols in frequency-time controlled high speed networks," *IEEE Transactions on Communications*, vol. 36, no. 4, pp. 430–440, 1988.

[62] "Road vehicles — FlexRay communications system — Part 5: Electrical physical layer conformance test specification," International Organization for Standardization, Standard, Sep. 2020.

[63] A. Grzemba, *MOST, the automotive multimedia network*, F. V. GmbH, Ed. 2011.

[64] "Telecommunications and exchange between information technology systems — Requirements for local and metropolitan area networks — Part 3: Standard for Ethernet," International Organization for Standardization, Standard, Feb. 2021.

[65] "Road vehicles — in-vehicle ethernet — part 1: General information and definitions," International Organization for Standardization, Standard, Oct. 2020.

[66] "Road vehicles — in-vehicle ethernet — part 2: Common physical entity requirements," International Organization for Standardization, Standard, Oct. 2020.

[67] H.-G. Hegering, A. Lapple, and S. S. Wilson, *Ethernet : building a communications infrastructure / Heinz-Gerd Hegering, Alfred Lapple ; [translated from the German by Stephen S. Wilson]*. Addison-Wesley Wokingham, England, 1993, viii, 389 p. :

[68] K. Matheus and T. Königseder, *Automotive Ethernet*, 1st. USA: Cambridge University Press, 2015.

[69] Markets and Markets. "Automotive communication technology market by bus module (lin, can, flexray, most, and ethernet), application (powertrain, body control & comfort, infotainment & communication, and safety & adas), vehicle class, and region - global forecast to 2025." (), [Online]. Available: `https://www.marketsandmarkets.com/Market-Reports/automotive-communication-technology-market-52588447.html` (visited on 01/05/2021).

[70] "Extended signal multiplexing in dbc databases," International Organization for Standardization, Application Note, 2019.

[71] CANEasy. "Multiplex messages." (), [Online]. Available: `https://www.caneasy.de/caneasyhelp/multiplex-botschaften.htm` (visited on 12/22/2021).

[72] CSS Electronics. "Clx000 intro, release fw 5.83." (2021), [Online]. Available: `https://canlogger.csselectronics.com/clx000-intro/CLX000Intro.pdf` (visited on 01/13/2021).

[73] KVASER AB. "Kvaser leaf light v2 user's guide." (2014), [Online]. Available: `https://www.kvaser.com/software/7330130980146/V2_0_0/kvaser_leaf_light_v2_usersguide.pdf` (visited on 11/16/2021).

[74] "Arduino." (), [Online]. Available: `https://www.arduino.cc/` (visited on 11/16/2021).

[75] "RaspberryPi." (), [Online]. Available: `https://www.raspberrypi.org/` (visited on 11/16/2021).

[76] "Road vehicles — communication between vehicle and external equipment for emissions-related diagnostics — part 3: Diagnostic connector and related electrical circuits: Specification and use," International Organization for Standardization, Standard, Apr. 2016.

[77] "Road vehicles — communication between vehicle and external equipment for emissions-related diagnostics — part 4: External test equipment," Standard, Feb. 2014.

[78] Baltijos automobilių diagnostikos sistemos. "Obd2 codes and meanings." (), [Online]. Available: `https://bads.lt/en/obd2-codes-and-meanings-2/` (visited on 11/16/2021).

[79] "Road vehicles — In-vehicle Ethernet — Part 1: General information and definitions," International Organization for Standardization, Standard, Feb. 2020.

[80] "Road vehicles — Diagnostic systems — Keyword Protocol 2000 — Part 3: Application layer," International Organization for Standardization, Standard, 1999.

[81] "Road vehicles — Diagnostic communication over Controller Area Network (DoCAN) — Part 2: Transport protocol and network layer services," International Organization for Standardization, Standard, Apr. 2016.

[82] "Overview of Functional Safety Measures in AUTOSAR," AUTOSAR, Standard, Nov. 2016.

[83] M. R. Moore, R. A. Bridges, F. L. Combs, and A. L. Anderson, "Data-driven extraction of vehicle states from can bus traffic for cyberprotection and safety," *IEEE Consumer Electronics Magazine*, vol. 8, no. 6, pp. 104–110, 2019.

[84] Vector. “Managing network and communication data with candb++.” (), [Online]. Available: `https://www.vector.com/int/en/products/products-a-z/software/candb/` (visited on 05/27/2020).

[85] E. J. Chikofsky and J. H. Cross, “Reverse engineering and design recovery: A taxonomy,” *IEEE Software*, vol. 7, no. 1, pp. 13–17, 1990.

[86] A. Goldsworthy, *The fall of Carthage : the Punic Wars, 265-146 BC*. Cassel Military Paperbacks, 2006.

[87] N. M. of the US Air Force. “Soviet union impounds and copies b-29.” (2015), [Online]. Available: `http://www.nationalmuseum.af.mil/factsheets/factsheet.asp?id=1852` (visited on 11/24/2021).

[88] V. Raja and K. Fernandes, *Reverse Engineering: A Industrial Perspective*. Springer-Verlag London Limited, 2008.

[89] A. F. Villaverde and R. J. Banga, “Reverse engineering and identification in systems biology: Strategies, perspectives and challenges,” 2013.

[90] I. Pyle, “Software reuse and reverse engineering in practice,” *Computing & Control Engineering Journal*, vol. 4, Jan. 1993.

[91] K. Wong, S. R. Tilley, H. A. Muller, and M.-A. Storey, “Structural redocumentation: A case study,” *IEEE Software*, vol. 12, no. 1, pp. 46–54, 1995.

[92] R. Koschke, “Software visualization in software maintenance, reverse engineering, and re-engineering: A research survey,” *Journal of Software Maintenance and Evolution: Research and Practice*, vol. 15, no. 2, pp. 87–109, 2003.

[93] Microsoft. “Use uml to reverse-engineer visual studio .net source code.” (2010), [Online]. Available: `https://support.microsoft.com/en-us/office/use-uml-to-reverse-engineer-visual-studio-net-source-code-0c4fc969-daa6-4169-a214-e9f554bbeaf7` (visited on 11/24/2021).

[94] C. Treude, F. Figueira Filho, M.-A. Storey, and M. Salois, “An exploratory study of software reverse engineering in a security context,” in *2011 18th Working Conference on Reverse Engineering*, IEEE, 2011, pp. 184–188.

[95] E. Eilam, *Reversing: Secrets of Reverse Engineering*. USA: John Wiley & Sons, Inc., 2005.

[96] E. Dupuy. “Java decompiler.” (), [Online]. Available: `http://java-decompiler.github.io/` (visited on 11/25/2021).

[97] Redgate. “.net reflector.” (), [Online]. Available: `https://www.red-gate.com/products/dotnet-development/reflector/` (visited on 11/25/2021).

[98] Hex-Rays. “Ida-pro.” (), [Online]. Available: `https://hex-rays.com/ida-pro/` (visited on 11/25/2021).

[99] B. D. Sija, Y.-H. Goo, K.-S. Shim, H. Hasanova, M.-S. Kim, and Z. Liu, “A survey of automatic protocol reverse engineering approaches, methods, and tools on the inputs and outputs view,” vol. 2018, 2018.

[100] Wireshark. “Wireshark - go deep.” (), [Online]. Available: https://www.wireshark.org/ (visited on 11/26/2021).

[101] J. Antunes, N. Neves, and P. Verissimo, “Reverse engineering of protocols from network traces,” in *2011 18th Working Conference on Reverse Engineering*, 2011, pp. 169–178.

[102] W. Cui, J. Kannan, and H. J. Wang, “Discoverer: Automatic protocol reverse engineering from network traces,” in *16th USENIX Security Symposium (USENIX Security 07)*, Boston, MA: USENIX Association, Aug. 2007. [Online]. Available: https://www.usenix.org/conference/16th-usenix-security-symposium/discoverer-automatic-protocol-reverse-engineering-network.

[103] R. Lin, O. Li, Q. Li, and Y. Liu, “Unknown network protocol classification method based on semi-supervised learning,” in *2015 IEEE International Conference on Computer and Communications (ICCC)*, 2015, pp. 300–308.

[104] A. Blin. “CAN bus reverse-engineering with Arduino and iOS.” (), [Online]. Available: https://medium.com/@alexandreblin/can-bus-reverse-engineering-with-arduino-and-ios-5627f2b1709a (visited on 08/10/2021).

[105] C. Smith. “The Car Hacker’s Handbook: A Guide for the Penetration Tester.” (), [Online]. Available: https://publicism.info/engineering/penetration/6.html (visited on 08/10/2021).

[106] C. Young, J. Svoboda, and J. Zambreno, “Towards reverse engineering controller area network messages using machine learning,” in *2020 IEEE 6th World Forum on Internet of Things (WF-IoT)*, IEEE, 2020, pp. 1–6.

[107] B. C. Nolan, S. Graham, B. Mullins, and C. S. Kabban, “Unsupervised time series extraction from controller area network payloads,” in *IEEE 88th Vehicular Technology Conference (VTC-Fall)*, IEEE, 2018, pp. 1–5.

[108] W. H. Day and H. Edelsbrunner, “Efficient algorithms for agglomerative hierarchical clustering methods,” *Journal of classification*, vol. 1, no. 1, pp. 7–24, 1984.

[109] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, “A density-based algorithm for discovering clusters in large spatial databases with noise.,” in *Kdd*, vol. 96, 1996, pp. 226–231.

[110] E. B. Fowlkes and C. L. Mallows, “A method for comparing two hierarchical clusterings,” *Journal of the American Statistical Association*, vol. 78, no. 383, pp. 553–569, 1983.

[111] S.-L. Developers. “Support vector machines.” (2021), [Online]. Available: https://scikit-learn.org/stable/modules/svm.html.

[112] ——, “Naive bayes.” (2021), [Online]. Available: https://scikit-learn.org/stable/modules/naive_bayes.html.

[113] ——, “Randomforestclassifier.” (2021), [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html.

[114] ——, “Neural network models (supervised).” (2021), [Online]. Available: https://scikit-learn.org/stable/modules/neural_networks_supervised.html.

[115] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, “Smote: Synthetic minority over-sampling technique,” *Journal of Artificial Intelligence Research*, vol. 16, pp. 321–357, 2002.

[116] P. D. Team. “Optimization with pulp.” (2009), [Online]. Available: https://coin-or.github.io/pulp/.

[117] DENSO Corporation. (), [Online]. Available: https://densoautoparts.com/ (visited on 01/13/2021).

[118] D. Wackerly, W. Mendenhall, and R. L. Scheaffer, *Mathematical statistics with applications*. Cengage Learning, 2014.

[119] T. Eiter and H. Mannila, “Computing discrete fréchet distance,” Citeseer, Tech. Rep., 1994.

[120] M. Ankerst, M. M. Breunig, H.-P. Kriegel, and J. Sander, “Optics: Ordering points to identify the clustering structure,” *ACM Sigmod record*, vol. 28, no. 2, pp. 49–60, 1999.

[121] S.-L. Developers. “Dbscan.” (2021), [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html.

[122] ——, “Optics.” (2021), [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.cluster.OPTICS.html.

[123] M. N. Azadani and A. Boukerche, “Performance evaluation of driving behavior identification models through can-bus data,” in *2020 IEEE Wireless Communications and Networking Conference (WCNC)*, IEEE, 2020, pp. 1–6.

[124] U. Fugiglando, P. Santi, S. Milardo, K. Abida, and C. Ratti, “Characterizing the" driver dna" through can bus data analysis,” in *Proceedings of the 2nd ACM International Workshop on Smart, Autonomous, and Connected Vehicular Systems and Services*, 2017, pp. 37–41.

[125] A. Cura, H. Küçük, E. Ergen, and İ. B. Öksüzoğlu, “Driver profiling using long short term memory (lstm) and convolutional neural network (cnn) methods,” *IEEE Transactions on Intelligent Transportation Systems*, 2020.

[126] K. Koscher, A. Czeskis, F. Roesner, *et al.*, “Experimental security analysis of a modern automobile,” in *2010 IEEE Symposium on Security and Privacy*, 2010, pp. 447–462.

[127] A. Palanca, E. Evenchick, F. Maggi, and S. Zanero, “A stealth, selective, link-layer denial-of-service attack against automotive networks,” in *Detection of Intrusions and Malware, and Vulnerability Assessment*, M. Polychronakis and M. Meier, Eds., Cham: Springer International Publishing, 2017, pp. 185–206.

[128] S. Checkoway, D. McCoy, B. Kantor, *et al.*, “Comprehensive experimental analyses of automotive attack surfaces,” in *USENIX Security Symposium*, San Francisco, vol. 4, 2011, pp. 447–462.

[129] 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, 2014.

[130] Wired. “Hackers Cut a Corvette’s Brakes Via a Common Car Gadget.” (2015), [Online]. Available: `https://www.wired.com/2015/08/hackers-cut-corvettes-brakes-via-common-car-gadget/` (visited on 12/01/2021).

[131] F. B. of Investigation. “Motor vehicles increasingly vulnerable to remote exploits.” (2016), [Online]. Available: `https://www.ic3.gov/Media/Y2016/PSA160317` (visited on 11/26/2021).

[132] R. Kammerer, B. Frömel, and A. Wasicek, “Enhancing security in can systems using a star coupling router,” in *7th IEEE International Symposium on Industrial Embedded Systems (SIES’12)*, IEEE, 2012, pp. 237–246.

[133] M. Bozdal, M. Samie, S. Aslam, and I. Jennions, “Evaluation of can bus security challenges,” *Sensors*, vol. 20, no. 8, 2020. [Online]. Available: `https://www.mdpi.com/1424-8220/20/8/2364`.

[134] A. Hanacek and M. Sysel, “Design and implementation of an integrated system with secure encrypted data transmission,” in *Computer Science On-line Conference*, Springer, 2016, pp. 217–224.

[135] T. P. Doan and S. Ganesan, “Can crypto fpga chip to secure data transmitted through can fd bus using aes-128 and sha-1 algorithms with a symmetric key,” SAE Technical Paper, Tech. Rep., 2017.

[136] A. S. Siddiqui, Y. Gui, J. Plusquellic, and F. Saqib, “Secure communication over canbus,” in *2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS)*, IEEE, 2017, pp. 1264–1267.

[137] W. A. Farag, “Cantrack: Enhancing automotive can bus security using intuitive encryption algorithms,” in *2017 7th International Conference on Modeling, Simulation, and Applied Optimization (ICMSAO)*, IEEE, 2017, pp. 1–5.

[138] M. Jukl and J. Cupera, “Using of tiny encryption algorithm in can-bus communication,” *Research in Agricultural Engineering*, vol. 62, no. 2, pp. 50–55, 2016.

[139] J. Yeom and S. Seo, “A methodology of can communication encryption using a shuffling algorithm,” in *2020 International Conference on Connected and Autonomous Driving (MetroCAD)*, IEEE, 2020, pp. 34–38.

[140] B. Groza, S. Murvay, A. Van Herrewege, and I. Verbauwhede, “Libra-can: A lightweight broadcast authentication protocol for controller area networks,” in *International Conference on Cryptology and Network Security*, Springer, 2012, pp. 185–200.

[141] 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.

[142] S. Nürnberger and C. Rossow, "Vatican–vetted, authenticated can bus," in *International Conference on Cryptographic Hardware and Embedded Systems*, Springer, 2016, pp. 225–237.

[143] 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.

[144] J. Van Bulck, J. T. Mühlberg, and F. Piessens, "Vulcan: Efficient component authentication and software isolation for automotive control networks," in *Proceedings of the 33rd Annual Computer Security Applications Conference*, 2017, pp. 225–237.

[145] Autosar. "Specification of Secure Onboard Communication." (), [Online]. Available: `https://www.autosar.org/fileadmin/user_upload/standards/classic/20-11/AUTOSAR_SWS_SecureOnboardCommunication.pdf` (visited on 11/30/2020).

[146] 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.

[147] 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.

[148] Y. Hamada, M. Inoue, H. Ueda, Y. Miyashita, and Y. Hata, "Anomaly-based intrusion detection using the density estimation of reception cycle periods for in-vehicle networks," *SAE International Journal of Transportation Cybersecurity and Privacy*, vol. 1, no. 11-01-01-0003, pp. 39–56, 2018.

[149] H. Lee, S. H. Jeong, and H. K. Kim, "Otids: A novel intrusion detection system for in-vehicle network by using remote frame," in *2017 15th Annual Conference on Privacy, Security and Trust (PST)*, IEEE, 2017, pp. 57–5709.

[150] E. Seo, H. M. Song, and H. K. Kim, "Gids: Gan based intrusion detection system for in-vehicle network," in *2018 16th Annual Conference on Privacy, Security and Trust (PST)*, IEEE, 2018, pp. 1–6.

[151] 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, 2018.

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

[153] W. Si, D. Starobinski, and M. Laifenfeld, "Protocol-compliant dos attacks on can: Demonstration and mitigation," in *2016 IEEE 84th vehicular technology conference (VTC-Fall)*, IEEE, 2016, pp. 1–7.

[154] C. Geng, S. Huang, and S. Chen, "Recent advances in open set recognition: A survey," *CoRR*, vol. abs/1811.08581, 2018.

[155] P. R. M. Júnior, R. Souza, R. de Oliveira Werneck, *et al.*, "Nearest neighbors distance ratio open-set classifier," *Machine Learning*, vol. 106, pp. 359–386, 2016.

[156] T. K. Ho, "Random decision forests," in *Proceedings of 3rd International Conference on Document Analysis and Recognition*, vol. 1, 1995, 278–282 vol.1.

[157] M. Z. Alom, T. M. Taha, C. Yakopcic, *et al.*, "A state-of-the-art survey on deep learning theory and architectures," *Electronics*, vol. 8, no. 3, 2019.

[158] F. Pedregosa, G. Varoquaux, A. Gramfort, *et al.*, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[159] F. Chollet *et al.* "Keras." (2015), [Online]. Available: `https://github.com/fchollet/keras`.

[160] A. Bendale and T. E. Boult, "Towards open set deep networks," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 1563–1572.

[161] L. P. Jain, W. J. Scheirer, and T. E. Boult, "Multi-class open set recognition using probability of inclusion," in *Computer Vision – ECCV 2014*, D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, Eds., Springer International Publishing, 2014, pp. 393–409.

[162] E. M. Rudd, L. P. Jain, W. J. Scheirer, and T. E. Boult, "The extreme value machine," *CoRR*, vol. abs/1506.06112, 2015. [Online]. Available: `http://arxiv.org/abs/1506.06112`.

[163] S. Kotz and S. Nadarajah, *Extreme value distributions: theory and applications*, W. Scientific, Ed. 2000.

[164] A. Niculescu-Mizil and R. Caruana, "Predicting good probabilities with supervised learning," in *Proceedings of the 22nd International Conference on Machine Learning*, ser. ICML '05, Bonn, Germany: Association for Computing Machinery, 2005, pp. 625–632.

[165] M. Liberatore and B. N. Levine, "Inferring the source of encrypted http connections," ser. CCS '06, Alexandria, Virginia, USA: Association for Computing Machinery, 2006, pp. 255–263.

[166] C. V. Wright, S. E. Coull, and F. Monrose, "Traffic morphing: An efficient defense against statistical traffic analysis.," in *NDSS*, Citeseer, vol. 9, 2009.

[167] T. O. Router. "Tor 0.3.1.7." (2017), [Online]. Available: `https://blog.torproject.org/tor-0317-now-released` (visited on 10/18/2021).

[168] K. P. Dyer, S. E. Coull, T. Ristenpart, and T. Shrimpton, "Peek-a-boo, i still see you: Why efficient traffic analysis countermeasures fail," in *2012 IEEE Symposium on Security and Privacy*, 2012, pp. 332–346.

[169] X. Cai, R. Nithyanand, T. Wang, R. Johnson, and I. Goldberg, “A systematic approach to developing and evaluating website fingerprinting defenses,” in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS ’14, Scottsdale, Arizona, USA: Association for Computing Machinery, 2014, pp. 227–238.

[170] L. Chaddad, A. Chehab, I. H. Elhajj, and A. Kayssi, “App traffic mutation: Toward defending against mobile statistical traffic analysis,” in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, IEEE, 2018, pp. 27–32.

[171] I. Hafeez, M. Antikainen, and S. Tarkoma, “Protecting iot-environments against traffic analysis attacks with traffic morphing,” in *IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, IEEE, 2019, pp. 196–201.

[172] N. Msadek, R. Soua, and T. Engel, “Iot device fingerprinting: Machine learning based encrypted traffic analysis,” in *IEEE wireless communications and networking conference (WCNC)*, IEEE, 2019, pp. 1–8.

[173] G. Sadanand. “Motorcycle ecu (engine control unit) explained.” (2019), [Online]. Available: `https://www.bikedekho.com/news/motorcycle-ecu-engine-control-unit-explained` (visited on 01/04/2022).

[174] L. van Dijk. “Future vehicle networks and ecus: Architecture and technology considerations.” (2017), [Online]. Available: `https://www.nxp.com/docs/en/white-paper/FVNECUA4WP.pdf` (visited on 01/04/2022).

[175] G. C. Congress. “Ihs markit: Sales of automotive ecus to hit $211b in 2030, 5% cagr.” (2017), [Online]. Available: `https://www.nxp.com/docs/en/white-paper/FVNECUA4WP.pdf` (visited on 01/04/2022).

[176] C. Electronics. “Canedge2: 2x can bus data logger (sd + wifi).” (), [Online]. Available: `https://www.csselectronics.com/products/can-bus-data-logger-wifi-canedge2` (visited on 11/25/2021).

[177] C. Quigley, D. Charles, and R. McLaughlin, “CAN Bus Message Electrical Signatures for Automotive Reverse Engineering, Bench Marking and Rogue ECU Detection,” in *SAE Technical Paper*, SAE International, Apr. 2019.

LIST OF ABBREVIATIONS

ACK	Acknowledge	17
ACTT	Automatic CAN Tokenization and Translation	38
AD	Anomaly Detection	37
AES	Advanced Encryption Standard	110
AHC	Agglomerative Hierarchical Clustering	44
AKS	Accuracy for KKC	118
AST	Abstract Syntax Tree	46
AUS	Accuracy for UUC	118
BE	Big Endian	22
BFC	Bit Flip Count	39
BFR	Bit Flip Rate	36
CAN	Controller Area Network	5
CANH	CAN High	15
CANL	CAN Low	15
CAN FD	Controller Area Network Flexible Data-Rate	14
CAV	Connected and Automated Vehicle	8
CE	correctly extracted tokens	38
CNN	Convolutional Neural Network	44
CRC	Cyclic Redundancy Check	17
CSI	Critical Signals Identifier	10
CSMA/CA	Carrier-Sense Multiple Access with Collision Avoidance	18
CSMA/CD	Carrier-Sense Multiple Access with Collision Detection	20
CV	Coefficient of Variation	73
DBC	Database CAN	28
DBSCAN	Density-Based Spatial Clustering of Applications with Noise	9
DES	Data Encryption Standard	110
DFD	Discrete Frechet Distance	xi
DID	Diagnostic ID	38
DLC	Data Length Code	17

DoCAN	Diagnostic communication over Controller Area Network	27
DTW	Dynamic Time Warping	92
DVT	Driven Vehicles Traces	57
ECU	Electronic Control Unit	5
EOF	End of Frame	17
EV	Electric Vehicle	46
EVM	Extreme Value Machine	119
EVT	Extreme Value Theory	120
FCNN	Fully Connected Neural Network	119
FD	Flexible Data-Rate	17
FDA	Format Decoding Accuracy	97
FG	Foreground	80
FM-Score	Fowlkes-Mallows score	44
FPR	False Positive Rate	68
FSM	Finite State Machine	33
FTDMA	Frequency-Time Division Multiple Access	19
GPS	Global Positioning System	10
HMM	Hidden Markov Model	44
IDE	Identifier extension bit	16
IDS	Intrusion Detection System	111
IEEE	Institute of Electrical and Electronics Engineers	20
IFS	Inter-frame spacing	17
ILP	Integer Linear Programming	10
IMU	Inertial Measurement Unit	10
IoT	Internet of Things	127
ISO	International Organization for Standardization	16
IVI	In-Vehicle Infotainment	7
KKC	Known Known Class	118
KUC	Known Unknown Class	118

KWP	Keyword Protocol	27
LE	Little Endian	22
LIN	Local Interconnect Network	5
LSB	Less Significant Bit	
LSTM	Long Short-Term Memory	43
ML	Machine Learning	7
MLP	Multilayer Perceptron	77
MOST	Media Oriented Systems Transport	5
MSB	Most Significant Bit	22
NB	Naive Bayes classifier	77
NN	Neural Network	9
NRMSE	Normalized Root Mean Squared Error	68
OBD-II	On-Board Diagnostics	8
ODVT	OpenDBC DVT	57
OEM	Original Equipment Manufacturer	6
OPTICS	Ordering Points To Identify the Clustering Structure	94
OSI	Open Systems Interconnection	14
OSR	Open Set Recognition	9
PID	Parameters ID	27
PVT	Parked Vehicles Traces	90
READ	Reverse Engineering of Automotive Data frames	10
RF	Random Forest	9
RMSE	Root Mean Squared Error	92
RPM	revolutions per minute	35
RTR	Remote Transmission Request	16
RSU	Roadside Unit	xi
SAE	Society of Automotive Engineers	25
SDDBC	Set of Derived DBC	29
SecOC	Secure Onboard Communication	111

SID	Service ID	27
SMOTE	Synthetic Minority Over-sampling TEchnique	9
SVM	Support Vector Machine	9
TANG	Transition Aggregated N-Grams	39
TCU	Telematic Control Unit	7
TDBC	total number of signals in the DBC file	40
TDM	Time Division Multiplexing	19
TDMA	Time Division Multiple Access	19
TE	total extracted signals	40
Tor	The onion router	127
UDS	Unified Diagnostic Services	27
UI	User Interface	47
UKC	Unknown Known Class	118
UML	Universal Modeling Language	32
UUC	Unknown Unknown Class	118
V2X	Vehicle-to-Everything	109
VANET	Vehicular Ad Hoc Network	8