

# A Survey and Comparative Analysis of Security Properties of CAN Authentication Protocols

Alessandro Lotto , Francesco Marchiori , *Student Member, IEEE*,  
Alessandro Brighente , *Member, IEEE*, and Mauro Conti , *Fellow, IEEE*

**Abstract**—The large number of Electronic Control Units (ECUs) mounted on modern cars and their expansive communication capabilities create a substantial attack surface for potential exploitation. Despite the evolution of automotive technology, the continued use of the originally insecure Controller Area Network (CAN) bus leaves in-vehicle communications inherently non-secure. In response to the absence of standardized authentication protocols within the automotive domain, researchers propose diverse solutions, each with unique strengths and vulnerabilities. However, the continuous influx of new protocols and potential oversights in meeting security requirements and essential operational features further complicate the implementability of these protocols.

This paper comprehensively reviews and compares the 15 most prominent authentication protocols for the CAN bus. Our analysis emphasizes their strengths and weaknesses, evaluating their alignment with critical security requirements for automotive authentication. Additionally, we evaluate protocols based on essential operational criteria that contribute to ease of implementation in predefined infrastructures, enhancing overall reliability and reducing the probability of successful attacks. Our study reveals a prevalent focus on defending against external attackers in existing protocols, exposing vulnerabilities to internal threats. Notably, authentication protocols employing hash chains, Mixed Message Authentication Codes, and asymmetric encryption techniques emerge as the most effective approaches. Through our comparative study, we classify the considered protocols based on their security attributes and suitability for implementation, providing valuable insights for future developments in the field.

**Index Terms**—Controller Area Network, Authentication, Vehicle Security, Comparative Analysis.

## I. INTRODUCTION

THE rising number of vehicles has led to increased traffic congestion and safety concerns for drivers [1], [2]. To address these challenges, industries, and academia are actively developing technologies for Intelligent Transportation Systems (ITSs). These systems aim to enhance road coordination through innovative services, including traffic management and vehicle-to-infrastructure communication [3], [4], [5]. While Vehicular Ad-hoc Networks (VANETs) are seen as a step towards secure ITSs and autonomous driving, it is crucial

to ensure security at individual network nodes [6]. In this context, securing the in-vehicle communication is paramount, considering the potential impact on network reliability and user safety [6]. Moreover, the automotive market's increasing focus on advanced technology has led to a proliferation of electronic components, such as Electronic Control Units (ECUs), that facilitate communication between vehicles [6].

The Controller Area Network (CAN), introduced by Robert Bosch GmbH in 1983, has become a renowned and widely used serial bus, initially designed for automotive applications [7], [8], [9]. Still, CAN has found versatile use in fields like medical devices, industrial automation, and robotics, thanks to its reliable, robust, efficient, and flexible communication capabilities [9], [10], [11]. Despite its adoption as the *de-facto* standard in modern cars, CAN lacks inherent security features, as its design assumes an isolated and friendly environment [12], [11], [13], [14]. However, with modern cars featuring various communication channels like Bluetooth, On-Board Diagnostic (OBD) ports, and cellular technology, interactive systems expose potential vulnerability surfaces for attackers [15], [16], [17]. The absence of security features in CAN, including confidentiality, encryption, data integrity, and authenticity, renders the system susceptible to exploitation. This vulnerability was demonstrated by C. Miller and C. Valasek, who remotely hacked a Fiat Chrysler (Jeep) through the internet-connected “Uconnect”, exploiting open Transmission Control Protocol (TCP) ports on the Sprint mobile network [18], [19], [20], [21]. This breach enabled remote code execution, providing attackers full control over the vehicle and leaving users powerless to regain control.

A crucial yet vulnerable aspect of CAN is its broadcast communication, where each message contains a field specifying data type rather than the sender's identity. Such a lack of sender identification poses severe authentication issues and security threats, endangering vehicle and passenger safety [22], [23], [24]. Researchers have proposed various authentication schemes for securing CAN communications to address this vulnerability. Despite widespread recognition of the authentication issue in the automotive industry and the proposal of several solutions, a practical lack of adoption persists in modern cars. This gap exposes vehicles to potential malicious attacks, such as the injection of packets in the CAN bus for vehicle theft [25]. This further highlights the need for comprehensive analysis and comparison of authentication solutions to understand the reasons behind their limited usage [26]. Moreover, given the constant introduction of new protocol proposals from researchers, who can often oversee

Authors are with the Department of Mathematics, University of Padova, via Trieste 63, Padova, Italy (email: alessandro.lotto@math.unipd.it, francesco.marchiori@math.unipd.it, alessandro.brighente@unipd.it, mauro.conti@unipd.it).

M. Conti is also affiliated with the Faculty of Electrical Engineering, Mathematics and Computer Science, Delft University of Technology, Mekelweg 4, 2628 CD Delft, Netherlands.

This work has been submitted to the IEEE for possible publication. Copyright may be transferred without notice, after which this version may no longer be accessible.

essential security requirements or necessary features for the protocols' implementation, the available body of literature has become particularly cumbersome and convoluted. This emphasizes the importance of thoroughly assessing the most promising authentication solutions in practical scenarios.

**Contributions.** In this paper, we provide a comparative analysis of the 15 most prominent authentication protocols for CAN. On top of the protocols' descriptions, we offer a comprehensive review and a unified model for comparing these protocols. Furthermore, we discern the critical security requirements necessary for the automotive framework, many of which have been overlooked or defined with limitations in related works. We bridge crucial gaps in the existing knowledge by highlighting these limitations and discussing how to address them. To complement our analysis, we introduce additional comparison criteria that can aid practitioners in selecting the most suitable protocol for real-world implementations. Lastly, we take a forward-looking approach by proposing enhancements that can fortify the existing protocols, making them more robust and secure. Additionally, we offer valuable insights for developing future authentication protocols in the context of the CAN bus. Together, these contributions advance our understanding of CAN authentication protocols and provide practical guidance for their implementation and improvement in automotive systems. To the best of our knowledge, this survey work marks the first of its kind in the literature, as also demonstrated in Table I.

The main contributions of our work are summarized as follows:

- We present a technical and detailed overview of **15 cryptographic authentication protocols** for CAN. To the best of our knowledge, this is the first contribution to provide a detailed review and unified model for comparison. Table II reviews the considered protocols, which are the most surveyed in the literature, with the addition of the two latest protocols.
- We identify and define **security criteria** necessary for CAN authentication protocols applied to the automotive framework. Some are either not considered in related works or defined with some limitations. We highlight such limitations and provide a discussion to address them.
- We identify and define **operational criteria** crucial for selecting the protocol for real-world implementation. In conjunction with security requirements, these characteristics provide a comprehensive analytical overview of the protocols.
- We propose **enhancements** that may be applied to make protocols more robust and secure and provide indications for developing future authentication protocols in the CAN bus.

**Organization.** The paper is organized as follows. Section II presents the related works. Section III provides an overview of CAN specifications and the current standard scenario. Section V presents the possible attacks to the CAN bus and defines the adversarial model. Section IV discusses hardware security solutions to introduce cryptographic primitives in CAN. Section VI presents all the authentication protocols

considered in this work. Section VII defines the criteria according to which the protocol comparison will be based. In Section VIII and Section IX the comparison is presented. Section X presents an analysis of the results from the comparison, and Section XI provides the takeaway messages. Finally, Section XII concludes the paper. Furthermore, Appendix A provides the necessary background knowledge about cryptographic elements that will be introduced in the paper.

## II. RELATED WORKS

Our research shows that most surveys about CAN security mainly analyze CAN vulnerability surfaces, possible exploits, and corresponding countermeasure approaches. Furthermore, the large part of existing surveys in the literature that focus on a specific security aspect, thus carrying out a detailed review and comparison of existing security mechanisms, mainly cares about Intrusion Detection System (IDS). On the contrary, only a few works specifically survey cryptographic authentication solutions in an extensive and in-depth manner. The absence of a comprehensive literature review about the authentication framework for the CAN bus and limitations shown by existing works prevents us from reaching a definite solution for CAN authentication protocols, which highlights the importance of conducting a comprehensive and in-depth comparative analysis between the most discussed and promising authentication protocols for CAN.

Several survey works overview the most common and discussed vulnerability surfaces and corresponding threats and exploits, together with possible countermeasure approaches for security solutions [27], [28], [29], [31], [34], [35], [38]. From the analysis of these surveys, we can conclude that the current state-of-the-art approaches to secure the CAN bus environment are encryption-based mechanisms (authentication and encryption schemes), IDSs, firewalls, and network segmentation. However, these papers only conduct a high-level and general discussion about possible countermeasure approaches and mitigation solutions, giving some case studies as examples. Works [34] and [38] present a dedicated section about authentication schemes. However, the protocol analysis and (when present) comparison are carried out from a high-level perspective without providing detailed security analysis and comparison criteria, which is our work's goal. These works are considerably noteworthy in value and thus will constitute the starting point for our work. Table I compares related works to ours, highlighting the contribution of our survey. Furthermore, Table II compares the protocols considered in this survey to the ones considered in the related works. We consider the most promising protocols from the literature and related survey works for our analysis.

*Nowdehi et al.* [30] conducted an effective comparison analysis between 10 authentication protocols for CAN. The authors defined five industrial requirements they identified as necessary to satisfy for a security solution to be usable in practice. Protocol comparison is performed in light of the defined requirements: effectiveness, backward compatibility, support for vehicle repair and maintenance, sufficient implementation details, and acceptable overhead. The limitations of their survey paper encompass concerns regarding cost-effectiveness in

TABLE I: Comparison of the surveys' contributions.

Survey	Year	Protocols Description	Protocols Comparison	Security Criteria	Operational Criteria	Security Analysis	Security Classification
Wolf et al. [27]	2004	○	○	○	○	○	○
Studnia et al. [28]	2013	○	○	○	○	○	○
Liu et al. [29]	2017	○	○	○	○	○	○
Nowdehi et al. [30]	2017	◐	●	○	●	○	●
Avatefipour et al. [31]	2018	◐	○	○	○	○	○
Groza et al. [32]	2018	◐	○	○	○	○	○
Gmiden et al. [33]	2019	◐	●	◐	◐	○	○
Bozdal et al. [34]	2020	○	○	○	○	○	○
Hartzell et al. [35]	2020	○	○	○	○	○	○
Aliwa et al. [36]	2021	◐	●	◐	◐	○	○
Jo et al. [37]	2021	◐	○	○	○	○	○
Fakhfakh et al. [38]	2022	◐	○	○	○	○	○
<b>This work</b>	<b>2023</b>	●	●	●	●	●	●

● indicates full consideration of a specific aspect.

◐ indicates partial consideration of a specific aspect (i.e., less protocols, security requirements, or complementary features).

○ indicates no consideration of a specific aspect.

mandating hardware-supported cryptographic primitives for all ECUs in safety-critical real-time systems, potential backward compatibility issues, and the necessity of acceptable overhead for hardware-based solutions. We present a more detailed analysis of these limitations in Section VII-A. It is worth noting that these limitations are rooted in a requirements definition that is implementation-oriented and performance-oriented, as stated by the authors. On the contrary, our paper prioritizes a security analysis of authentication protocols for CAN, marking it as the first of its kind.

*Groza et al.* [32] conducted a survey presenting the evolution of protocol proposals for CAN security in chronological order from 2007 to 2016. Besides, the authors discussed some cryptographic tools and solutions for the key-sharing problem when using encryption-based mechanisms in CAN. Some authentication protocol proposals were briefly mentioned as examples of possible authentication approaches, but no detailed description nor protocol analysis were given. Furthermore, no comparison between the protocols was presented.

*Gmiden et al.* [33] work surveys IDS and cryptographic security solutions designed for the CAN bus. Regarding the cryptographic solutions, the authors evaluated and compared the considered protocols according to the following criteria: authentication, integrity, confidentiality, backward compatibility, replay attack resistance, and real-time performance. The survey's limitations include a narrow focus on only six protocols, which does not comprehensively represent the entirety of the literature. Moreover, the lack of precise definitions for evaluation criteria, particularly regarding backward compatibility, raises ambiguity about protocol compliance. The generic nature of the protocol evaluation in the survey results in a limited and high-level analysis, potentially overlooking variations in authentication effectiveness in different scenarios and against different types of attackers.

*Aliwa et al.* [36] presented the CAN-bus protocol and its limitations with related vulnerabilities and corresponding exploits an attacker can carry out. Then, they discussed possible approaches to enforce security in the CAN framework - such as authentication, encryption, and IDS deployment -

and presented some case studies. The authors outlined several protocols for the authentication framework according to their main features and working principles. Furthermore, they provided a summary table that compares the protocol based on several aspects comprising security and operational features. We highlight, however, that the considered comparison criteria are less than those we will consider for our analysis. On the other hand, this work lacks a precise definition of evaluation criteria and a proper security analysis and discussion.

*Jo et al.* [37] presented a classification of possible CAN attack surfaces (physical access-based, wireless access-based requiring initial physical access, wireless access-based without physical access) and a categorization of defense approaches (preventative protection, intrusion detection, authentication, and post-protection). For each defense category identified, the authors gave an overview of several implementation approaches, analyzing their pros and cons by presenting some implementation proposals in the literature. Regarding the authentication part, the authors focused on solutions for the authentication key sharing and transmission of the authentication tag. They revised possible approaches for each authentication aspect and pointed out some related works employing them. However, this survey did not define any protocol evaluation criteria, nor was any protocol comparison performed. Rather, a conceptual analysis of the authentication methodology, advantages, and drawbacks of the considered protocols was presented.

Finally, *Pesè et al.* [39] proposed the S2-CAN authentication protocol. In their work, the authors compared their proposal with other related works. However, the compared protocols were generally presented in their main authentication strategy, and the comparison was carried out according to technical details of the induced latency and authentication method, i.e., employed cryptographic algorithms, Message Authentication Code (MAC) length, use of hardware modules. However, no comprehensive and in-depth security analysis and comparison was present.

TABLE II: Comparison of the surveys' considered protocols for CAN authentication.

Survey	CANAuth	Car2X	LinAuth	MaCAN	LCap	CaCAN	YeCure	Woo-Auth	LeiA	vatiCAN	Libra-CAN	VulCAN	TOUCAN	AuthenticCAN	S2-CAN
Wolf et al. [27]	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
Studnia et al. [28]	○	●	○	●	○	○	○	○	○	○	●	○	○	○	○
Liu et al. [29]	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
Nowdehi et al. [30]	●	●	●	●	○	●	●	●	○	●	●	○	○	○	○
Avatefipour et al. [31]	○	○	●	○	○	○	●	○	○	○	○	○	○	○	○
Groza et al. [32]	●	○	○	●	○	●	○	●	○	○	●	○	○	○	○
Gmiden et al. [33]	○	○	○	○	○	●	●	●	○	○	●	●	○	○	○
Bozdal et al. [34]	○	○	○	○	○	○	●	○	○	○	●	○	○	○	○
Hartzell et al. [35]	○	○	●	○	○	○	○	○	○	○	○	○	○	○	○
Aliwa et al. [36]	●	○	●	○	●	●	●	○	●	○	○	○	●	○	○
Jo et al. [37]	○	○	○	○	○	●	●	●	●	●	○	●	●	○	○
Fakhfakh et al. [38]	○	○	○	○	○	○	●	●	○	○	○	○	○	○	○
<b>This work</b>	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●

### III. BACKGROUND

In this section, we provide a background overview of the CAN standard protocol [8], [9] in its main and most significant aspects necessary for a complete comprehension of the following discussions (Section III-A). Moreover, we also present two other protocols, CAN+ [40] (Section III-B) and CAN-FD [41] (Section III-C), which are two more recent versions of the standard CAN protocol.

#### A. CAN

In this section, we provide a presentation of the four core aspects of the CAN standard: (i) the physical layer and ECU structure, (ii) the structure of a CAN frame, (iii) the procedure to access to the bus, (iv) the arbitration algorithm.

1) *Physical Layer*: CAN is a multi-master differential two-bus protocol. Every ECU of the system is connected to all the others through two wires named CAN\_H and CAN\_L [8], [9]. Being a multi-master protocol, every message is broadcast to all the other ECUs. The differential property is related to the physical encoding of the bits into voltage values, which occurs as follows.

$$\begin{cases} 0 \rightarrow \text{CAN\_H} - \text{CAN\_L} \approx 2V, \\ 1 \rightarrow \text{CAN\_H} - \text{CAN\_L} \approx 0V. \end{cases}$$

In the equation, bit 0 is referred to as *dominant* and bit 1 as *recessive*. This labeling suggests the bus behavior when a 0 and 1 are written simultaneously: as a 2V difference dominates over a 0V difference, the bit 0 will always overwrite the bit 1. Moreover, thanks to this differential encoding, the electrical current flows in opposite directions for the two wires, resulting in a field-canceling effect. This makes CAN immune to noise and fault tolerant.

In practice, the signal is imperfect and needs some time to stabilize. Therefore, when writing and reading the bus, delays must be considered. The bit transmission time is then divided into 8 to 25 time intervals, and after about 2/3 of the bit time, the signal sampling is performed. A synchronization process at the beginning of the transmission is required as

well [40]. ECUs synchronize by themselves the position of the sample point to a specific phase in relation to the edges of the monitored bit stream. Receivers shift the phases of the samplings relatively to the phase of the transmitter's ones. Thus, a node's specific phase shift depends on the signal delay time from the transmitter to that node [41]. The typical speed of a CAN bus is 500 Kbit/s with a limit of 1 Mbit/s.

2) *ECU Structure*: A typical ECU usually comprises three parts: a *micro-processor*, a *CAN controller*, and a *CAN transceiver*. Even though messages are always broadcasted, the CAN controller filters out unwanted messages by looking at the identification field of the message, thus understanding if the content is of interest. CAN transceivers connect the physical transmission medium to the controller and physically operate the two bus lines to generate the logical bits.

3) *CAN Frame*: A CAN data frame is composed of several fields, and generally, there are two formats of the protocol: the standard and extended, with an 11-bit and 29-bit identifier, respectively [9]. An overview of these two formats is shown in Figure 1. Each field is described as follows.

- **Start of Frame (SOF)**: single dominant bit marking the beginning of the message.
- **Identifier (ID)**: 11-bit field that establishes the message's priority and discriminates the type of payload, i.e., the type of data sent. The extended version also uses the *Extended ID (IDE)* field.
- **Remote Transmission Request (RTR)**: dominant single-bit field when information is required from another node. In this case, the ID identifies the type of information requested.
- **Substitute Remote Request (SRR)**: substitutes the RTR bit as a placeholder in the extended version.
- **Identifier Extension (IDE)**: single bit field, dominant if Extended ID is used.
- **r0, r1**: unused, reserved for future standard definitions.
- **Data Length Code (DLC)**: 4-bit field indicating the number of bytes of the payload. Values are taken from 0000 to 1000.
- **Data**: message payload ranging from 1 to 8 bytes.

- **Cyclic Redundancy Check (CRC):** contains the checksum of the data field for error detection.
- **Acknowledge (ACK):** 2-bit field. The first bit is originally set to recessive. Every node receiving the message correctly overwrites it with a dominant bit. If the bit is left recessive, no ECU correctly receives the message, which is discarded and sent again after re-arbitration. The second bit is used as a delimiter.
- **End-of-frame (EOF):** 7 recessive bits marking the end of the frame.

S O F	ID	R T R	I D E	r0	DLC	Data	CRC	ACK	E O F
0	11 bits	0	0	0	4 bits	8 - 64 bits	16 bits	2 bits	1111111

(a) Standard CAN 11-bit identifier.

S O F	ID	S R R	I D E	Extended ID	R T R	r1	r0	DLC	Data	CRC	ACK	E O F
0	11 bits	0	0	18 bits	0	0	0	4 bits	8 - 64 bits	16 bits	2 bits	1111111

(b) Extended CAN 29-bit identifier.

Fig. 1: CAN frame structure.

4) **Arbitration Algorithm:** CAN is a Carrier-Sense Multiple-Access/Collision Detection and Arbitration Message Priority (CSMA/CD+AMP) communication protocol [42]. CSMA means that each node must wait for a determined period of inactivity before attempting to access the bus and send a message. CD+AMP means collisions with multiple messages sent simultaneously are solved through a bit-wise arbitration algorithm based on the message priority encoded in the identifier field. When two or more ECUs try to access the bus simultaneously, only the one with higher priority succeeds, and no other ECU attempts to access the bus until complete message transmission [43]. Besides, the arbitration algorithm also assures that no message is lost due to an interruption caused by a bus access attempt from an ECU with higher priority than the one currently transmitting. Notice that, thanks to the differential encoding, a higher priority corresponds to a lower ID, as bit 0 dominates over bit 1. As a drawback, the arbitration mechanism strongly limits CAN bus bandwidth. Indeed, protocol specifications require that the signal propagation between any couple of nodes is less than half of one-bit time [9]. This clearly defines an upper bound for the bit rate and (physical) bus length.

## B. CAN+

In recent years, the number of ECUs used is rising, and the workload on the bus is getting higher, causing a longer response time and degrading the real-time capability of the system. The CAN+ protocol was introduced in 2009 as a solution for higher throughput, enabling a data rate up to 16 times higher compared to the standard CAN [40]. By analyzing the transmission time of a single bit in classic CAN protocol, researchers found a *gray zone* period during which the bus could take any value without disturbing the normal CAN communication. This gray zone is delimited by the *synchronization* and *sampling zones*, as shown in Figure 2,

and it is the one used by CAN+ to transmit the additional information at a higher data rate. These additional bits are called *overclocked bits*. Since overclocking is possible only when we can assure the presence of a single transmitter, we can exchange the overclocked bits during data transmission only. By defining  $f$  as the *overclocking factor*, and recalling that a CAN standard data field is at most 8 bytes, the maximum amount of transmitted bytes per message is:

$$N_{total} = 64 + f \times 64 = (1 + f) \times 64.$$

A detailed analysis shows that the overclocking factor can take a maximum value of 16, over which no feasible overclocking is possible [40]. It is also worth mentioning that to implement the CAN+ protocol properly, transceivers need to support data rates up to 60 Mbit/s [40].

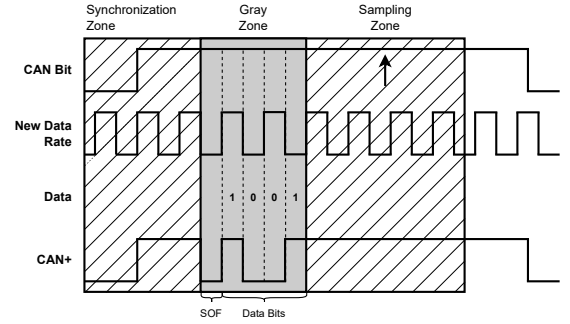


Fig. 2: Transmission of a standard CAN bit and of overclocking bits of CAN+.

## C. CAN-FD

CAN with Flexible Data rate (CAN-FD) was designed in 2011 to (i) improve the header-to-payload ratio, and (ii) speeding up the frame transmission [41]. The protocol offers major improvements to the standard CAN [44].

- Extension of Data field up to 64 bytes.
- Average transmission rate increased up to 6 Mbit/s.
- Higher performance in the CRC algorithm, lowering the risk of undetected errors.

As CAN+, CAN-FD exploits the fact that only a single node can access the bus after the arbitration. Thus, switching to a faster and predefined transmission rate is possible. It is worth noting that all nodes must switch to the new rate synchronously and return to the standard rate at the end of data transmission. The speed-up factor is thus determined by (i) the speed of the transceivers (if the bit time is too short, it will not be possible to decode it) and (ii) the time resolution of the synchronization mechanism. Other secondary precautions must be taken for complete backward compatibility and functioning of the protocol [41]. Besides data field extension, the protocol modifies the frame by defining new fields, as shown in Figure 3.

- **Flexible Data Format (FDF):** single dominant bit providing an edge for resynchronization before a possible bit rate switch.

- **Bit Rate Switch (BRS):** single bit that defines whether to switch into CAN-FD decoding mode. If recessive, ECUs must switch to the faster bit rate.
- **Error State Indicator (ESI):** single bit set to dominant for error active, recessive for error passive.

Furthermore, the length of the payload is still defined by the DLC field but with an additional encoding: values from 1001 to 1111 define the data length up to 64 bytes. Thus, each increase corresponds to a 4-byte increase in the data length (e.g., 1001 corresponds to 12 bytes, 1010 to 16 bytes).

S O F	ID	R T R	I D E	F D F	r0	B R S	E S I	DLC	Data	CRC	ACK	E O F
0	11 bits	0	0	0	0	1 bit	1 bit	4 bits	8 - 64 bits	16 bits	2 bits	1111111

(a) Standard CAN-FD 11-bit identifier.

S O F	ID	S R R	I D E	Extended ID	R T R	F D F	r1	r0	B R S	E S I	DLC	Data	CRC	ACK	E O F
0	11 bits	0	0	18 bits	0	0	0	0	1 bit	1 bit	4 bits	8 - 64 bits	16 bits	2 bits	1111111

(b) Extended CAN-FD 29-bit identifier.

Fig. 3: CAN frame structure.

#### IV. SYSTEM MODEL

As CAN protocol design was not meant for security, some complications arise when trying to build an authentication protocol on top of it. These obstacles are mainly related to the introduction of cryptographic primitives and the management of related security parameters. Besides, because CAN IDs only identify the message's content type when introducing cryptographic primitives, there may be a need for transmitter identification.

In this section, we address the following two main aspects: (i) handling cryptographic primitives (Section IV-A), and (ii) transmitter-receiver identification (Section IV-B).

##### A. Hardware Security Modules

Generally, using cryptographic primitives requires a secure storage of keys and related parameters to offer accountability and confidentiality [45]. Besides, when moving into the vehicular framework, we must consider other requirements and limitations, such as hard real-time constraints, power consumption, limited computing and storage resources, and implementation and maintenance costs. To meet these constraints and limitations, the European E-Safety Vehicle Intrusion Protected Applications (EVITA) project aimed to design building blocks for secure automotive on-board networks for securing security-relevant components [46]. As such, they designed Hardware Security Modules (HSMs) as *root of trust* modules to be easily integrated as an on-chip extension to the ECUs [47]. A HSM provides secure hardware and cryptographic functions for secure communication between nodes without occurring in high costs [48]. Also, engineers developed low-level drivers based on AUTOSAR to interact with EVITA HSMs [49]. AUTOSAR is a set of specifications and standards for Secure On-board Communication Modules to reduce costs and improve ECUs scalability [50].

Components of an HSM are divided into mandatory and optional, depending on the security requirements that must be fulfilled. EVITA also specifies three HSM variants to meet different security levels and cost-effectiveness.

- **Full HSM** – It focuses on protecting the in-vehicle domain from security vulnerabilities from vehicle-to-everything communications. It provides the maximum functionality level, security, and performance among the three HSM variants. It is designed to provide a security lifetime of over 20 years. It comprises blocks to perform symmetric and asymmetric cryptographic operations, such as hash computations and random number generations [49]. The major cryptographic building blocks are composed as follows.
  - *ECC-256-GF(p)*: high-performance asymmetric cryptographic engine based on a high-speed 256-bit elliptic curve arithmetic using NIST-approved prime field parameters [51].
  - *WHIRLPOOL*: AES-based hash function, as proposed by NIST [52].
  - *AES-128*: symmetric block encryption/decryption engine using the official NIST advanced encryption standard [53].
  - *AES-PRNG*: pseudo-random number generator, seeded with a truly random seed from a true internal physical random source.
  - *Counter*: a 64-bit monotonic counter function block that serves as a simple, secure clock alternative.
- **Medium HSM** – It aims to secure communications only within the vehicle. Compared to the full version, it is not provided with a hardware Error Correction Code (ECC) engine and a hardware hash engine. This module can quickly execute symmetric and non-time-critical asymmetric operations. Security credentials are permanently protected since they are kept from the application CPU [49].
- **Light HSM** – It focuses on protecting ECUs, sensors, and actuators. The security scheme is limited to a single very-specialized symmetric AES hardware accelerator, while the application CPU handles all security credentials. This fulfills the strict cost and efficiency requirements typical of sensors and actuators [49].

##### B. Transmitter-Receiver Identification

The other issue that naturally occurs when introducing cryptography on the ECUs into the CAN framework is identification. Indeed, several protocols we consider expect some direct communication between nodes (i.e., “node A sends to node B”). However, direct communication is not supported in a broadcast environment without sender/receiver identification. Hence, some enhancements to the standard are required. Since the ID field does not identify the sender or receiver but the payload data type, direct communications should require the sender and receiver identifiers to be attached to the frame. Without such information, the receiver could not identify the transmitter and the correct cryptographic key for decryption.

As a result, we need ECU-specific internal identifiers to provide such functionalities. Since not all the considered protocols mention this fact explicitly, we assume ECUs to be provided with internal specific identifiers when not directly reported.

One next step would be to consider embedding these identifiers into CAN frames. We identify using the extended CAN ID field as the optimal solution since it is very unlikely that all the  $2^{29}$  possible values are used for the standard CAN IDs. Therefore, available bits may be used for this purpose. This prevents adding further bits in frames, which would lead to an increase in the bus load. On the other hand, manufacturers can arbitrarily decide their specifications since there is no global standard to this scope yet. We report the existence of the *SAE J1939* protocol, widely adopted in industrial applications as a high-level protocol built on top of CAN, which also gives specifications on how to structure the extended ID field to include ECU-specific identifiers [54], [55], [56].

## V. THREAT MODEL

This section defines the adversarial model we consider for our analysis and investigates the possible attacks on an adversary may pursue on the CAN bus. Considering what has already been discussed in the related works, known limitations and vulnerabilities of the CAN protocol, we classify attacks in two classes: *injection* and *Denial of Service (DoS)* [12], [31], [39], [37].

The first class of attacks envisions the injection of malicious packets into the bus to modify the behavior or take control of the system. Among injection attacks, we distinguish between *replay attacks* and *masquerade attacks*. A replay attack consists of injecting previously recorded legitimate messages without modification. In a masquerade (or fabrication) attack, the attacker properly builds the content of the malicious packet to be transmitted, choosing the header and payload of the message. This attack involves sending messages with forged IDs, impersonating any other legitimate ECU authorized to send those specific IDs. It is worth noting that the replay attack is a special case of the masquerade attack. Still, we prefer to distinguish between the two since, in replay attacks, the attacker takes no action on the message apart from recording and re-transmitting it afterward.

DoS attacks consist of preventing legitimate ECUs (or a specific target ECU) from accessing the bus and transmitting messages. This happens because a malicious entity may flood the bus with high-priority messages, not allowing the transmission of frames with lower ID values. Consequently, a DoS attack prevents the correct functioning of the system. However, DoS attacks are out of the scope of this work as we cannot prevent them with cryptographic authentication mechanisms alone. Rather, we would need a higher perspective system, such as an IDS, capable of monitoring and analyzing the ongoing traffic in the bus and recognizing whether a DoS attack is currently running.

After defining the possible attacks on the CAN bus, we need to define how the attacker can perform an injection attack. To get access to the CAN bus, an attacker may (i) make use of an external device or (ii) compromise a legitimate ECU. In the

first case, we define an external device as any instrument not belonging to the original system. This can be either an external non-legitimate ECU physically plugged into the network or a diagnostic instrument attached to the OBD port through which it is possible to inject packets. In the second case, the adversary manages to compromise a legitimate ECU by physically tampering with the device or remotely hacking it. The two most common attacker behaviors are described as follows.

- *Target Injection* – Using the compromised ECU, the attacker sends packets with ID values the ECU is allowed to send, but faking the data field's values. We refer to this scenario as *target injection* because we do not classify it as a masquerade attack but as false data transmission. In this case, the attacker must tamper with exactly an ECU using the target IDs. We do not consider this scenario in our discussion since detecting and avoiding such attacks simply with an authentication scheme is impossible. It would be necessary, in fact, to have a specifically designed IDS.
- *Masquerade Injection* – In this scenario, the attacker exploits the tampered ECU to send messages with IDs it was not supposed to use, thus performing a masquerade or replay attack.

We assume the attacker can remotely compromise an ECU and perform a masquerade injection in all scenarios. We can justify our claim thanks to the fact that, in general, modern vehicles are equipped with an ECU that keeps remote access ports open for external networks such as WiFi or Bluetooth [37]. Besides, it is much harder to implement a target injection remotely than physically compromising the target ECU. However, this does not mean it is not possible.

Finally, we assume the following conditions to hold [36].

- **The attacker fully knows the underlying system.** The attacker knows if security mechanisms are in place and, in case, the specifics of the mechanism in use.
- **The attacker has full access to the bus.** The attacker can read and record exchanged messages. This assumption is reasonable given that potential attackers may have physical access to the CAN bus through multiple points within the vehicle, both internal and external, including locations like headlights [25].
- **It is not possible for the attacker to access data stored in protected memory from the outside.** The attacker cannot access the protected data of an ECU from another ECU or by physically tampering with the target device. Rather, they can access the protected data of the compromised ECU. This assumption is justified by the discussion in the next section introducing tamper-proof memories [57].

## VI. AUTHENTICATION PROTOCOLS

In this section, we present the protocols considered for our comparison. We provide a detailed description of their key working points, whereas information related to design decisions and performance results can be found, if not reported, in the referenced papers. Furthermore, in this section, we only

focus on the protocols' implementation explicitly stated in the papers. We instead present additional considerations in Section VIII and Section IX. Table III summarizes the protocols' authentication approaches and provides the organization of this section.

TABLE III: List of considered CAN authentication protocols and their descriptions.

Sec.	Protocol	Year	Standard	Authentication
VI-A	CANAuth [13]	2011	CAN+	HMAC
VI-B	Car2X [58]	2011	CAN	MAC
VI-C	LCAP [14]	2012	CAN	Hash chain
VI-D	LinAuth [59]	2012	CAN	MAC
VI-E	MaCAN [60]	2012	CAN	MAC
VI-F	CaCAN [61]	2014	CAN	Centralized HMAC
VI-G	VeCure [62]	2014	CAN	Trust-group MAC
VI-H	Woo-Auth [12]	2015	CAN	HMAC
VI-I	LeiA [7]	2016	CAN	Lightweight cryptography
VI-J	vatiCAN [63]	2016	CAN	HMAC
VI-K	LiBrA-CAN [64]	2017	CAN CAN+ CAN-FD	M-MAC
VI-L	VulCAN [65]	2017	CAN	MAC
VI-M	TOUCAN [66]	2019	CAN	Chaskey algorithm
VI-N	AuthentiCAN [67]	2020	CAN-FD	Asymmetric
VI-O	S2-CAN [39]	2021	CAN	Payload cycling shifting encoding

#### A. CANAuth

CANAuth (2011) bases its security upon Hash-based Message Authentication Codes (HMACs) and Session Keys [13]. Because of the limited payload size of standard frames and the hard real-time requirement for an in-vehicle communication protocol, the authors decided not to put inside (or attach to) the data frame of the HMAC nor to send a long data packet over multiple messages. Therefore, they decided to build the authentication mechanism on top of CAN+.

1) *Session Key Generation*: The key establishment procedure assumes that each node  $i$  possesses one or more 128-bit pre-shared master keys  $K_i$ . Also, the protocol introduces the concept of *Group messages*  $\mathcal{G}_i$ , where  $\mathcal{G}$  is the group and  $i$  is the group index. This allows the authentication of a group of related messages using the same key, called *Group key*. Keys are stored in a tamper-proof memory. The session key  $K_{S,i}$  for the group  $\mathcal{G}_i$  is generated by the first node that attempts to send message  $M_i \in \mathcal{G}_i$ . In case multiple nodes transmit  $M_i$ , the key established is the one generated by the node with the lowest ID. The session key  $K_{S,i}$  is derived as:

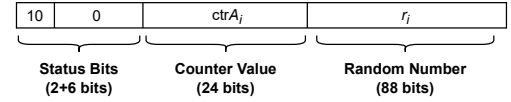
$$K_{S,i} = \text{HMAC}(K_i, \text{ctr}A_i \parallel r_i) \mod 2^{128},$$

where  $\text{ctr}A_i$  is a counter for message  $M_i$  stored in a non-volatile memory, and  $r_i$  is a random number. By applying the  $\text{mod} 2^{128}$  operation, only the 128 least significant bits of the HMAC are considered.

By knowing the counter and the random value, every node with the master key  $K_i$  can generate  $K_{S,i}$ . Since it is the first sender to define  $\text{ctr}A_i$  and  $r_i$ , it must also deliver them to all the other nodes. Hence, the session key establishment process develops into two phases, as shown in Figure 4.

- *Transmission of parameters*. The transmitter broadcasts a CAN message with attached a CAN+ message structured as in Figure 4a. The first bit of the status frame is recessive to signal that the key establishment phase is occurring. The second bit is set dominant to signal that this is the first of two messages. The remaining six bits are dominant and are unused.
- *Authentication of the key establishment*. The transmitter broadcasts a second message structured as shown in Figure 4b to prove the effective knowledge of the session key. In this case, both the first two bits are recessive. The payload of the message consists of a 112-bit signature

$$\text{sig}A_i = \text{HMAC}(K_{S,i}, \text{ctr}A_i \parallel r_i) \mod 2^{112}.$$



(a) Frame in the first part of session key establishment.



(b) Frame in the second part of session key establishment.

Fig. 4: CANAuth session key generation exchanged messages.

If any receiving ECUs raises an error in response to any of the two messages, the transmitting node will restart the transmission from the first message but change the counter (the random number could be the same as before).

2) *Message Authentication*: Once the session key establishment process has been completed, messages can be authenticated. The authentication of the CAN frame  $M_i$  is performed by transmitting a CAN+ message whose payload comprises a counter  $\text{ctr}M_i$ , used for replay attack resistance, and the signature  $\text{sig}M_i$ . An overview of the CANAuth frame used for authentication is shown in Figure 5.

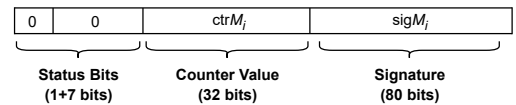


Fig. 5: CANAuth data frame during message authentication.

Notice that the counter  $\text{ctr}M_i$  is independent of  $\text{ctr}A_i$  (which is used for session key establishment), and it must be increased at least by one for every authenticated message. Receiving nodes accept the message if the received  $\text{ctr}M_i$  value exceeds the stored one. If so, they check for the correctness of the signature as follows:

$$\text{sig}M_i = \text{HMAC}(K_{S,i}, \text{ctr}M_i \parallel M_i) \mod 2^{80}.$$



### B. Car2X

Car2X (2011) is a cost-effective approach to ensuring trust in Car2X applications using in-vehicle symmetric cryptography. The protocol uses MAC-based authentication and allows MAC truncation. The security solution establishes trust between ECUs through an open and symmetric cryptography approach. It integrates into vehicles' existing infrastructure, requiring only the addition of security hardware modules essential for cryptographic acceleration.

1) *Architecture*: The security framework employs HSMs to store symmetric keys, ensuring secure and controlled key access. Metadata, including expiration dates and use-flags, allows for differentiated functionalities. The system uses dynamic key exchanges through a Key Master (KM), distributing keys for authentication and transport encryption. Multiple KMs support domain-specific key sharing. Integrated into a broader security framework, the system provides API abstractions for entity authentication, secure communication, and access control. The communication module facilitates flexible and secure communication channels, supporting various attributes and security levels through key exchanges and secure channel establishment managed by the access control framework and key distribution protocol.

2) *Protocol*: The key distribution and entity authentication process involves secure communication between an ECU (e.g.,  $e_1$ ) and a group of ECUs (e.g.,  $g_x$ ). A session key pair ( $k_{s,g}$ ,  $k_{s,v}$ ) is locally generated at  $e_1$ , with  $k_{s,v}$  being exportable. This key is exported as an encrypted key blob ( $b_1$ ) to the Key Master (KM) using the transport key ( $k_{1,t}$ ) and authenticated with  $k_{1,a}$ . These two keys are pre-shared with the KM. The KM authorizes communication and distributes the session key to group members, ensuring cryptographic operations occur within HSMs. The protocol incorporates security features, preventing impersonation and imposing a limited key lifetime of 48 hours. Indeed, thanks to HSM's use-flags, an attacker cannot impersonate a group's sender despite using symmetric keys. Due to constraints like limited message payload in CAN buses, a data segmentation approach is employed for transmitting cryptographic keys and signatures. This segmentation introduces delays and relies on packet sequencing to address transmission challenges. The protocol is augmented with a mandatory security header indicating cryptographic payload details to enhance security. Privacy and security requirements vary, prompting the option for MAC truncation for efficiency, considering factors like bus speed and load. A minimum 32-bit MAC length is deemed reasonable within specified environmental limitations, balancing security and resource constraints.

### C. LCAP

Lightweight CAN Authentication Protocol (LCAP) (2012) provides source authentication by using a hash chain mechanism [14]. For each pair of Sender-Receiver ECU ( $S, R$ ), there exists a pre-shared 128-bit secret key  $K_{SR}$  assumed to be stored in a protected memory. The authors then define *data messages*, the class corresponding to the standard CAN messages, and *handshake messages*, the class of messages

that identify a specific action. Within the handshake messages and for each sender-receiver pair, we distinguish between 5 actions, each with the same 11-bit ID field but different IDE value: *Channel Setup Request*, *First Response Message*, *Consecutive Response Message*, *Soft Synchronization Request*, *Hard Synchronization Request*. This implies that in addition to the standard CAN identifiers,  $5 \cdot n \cdot m$  identifiers must be added. The last two bytes of each message contain the checksum of the first six. The protocol can work in two possible operating modes: *extended mode*, where the hash value is sent using the extended identifier field, and *standard mode*, where the hash value is sent within the payload.

1) *Initialization*: Each sender creates the following elements.

- $H_C$  – Handshaking hash chain.
- $C_C$  – Channel hash chain.
- $K_S$  – 80-bit session key.
- $K_H$  – 16-bit HMAC key.
- $M_C^i$  – Hash chain for message type  $i$ . One chain is created for each message type  $i$  the ECU can send.

We use the subscript  $C$  to define the whole chain and the subscript  $j$  to refer to the  $j^{th}$  element of the chain.

2) *Channel Setup*: The sender delivers  $K_S$ ,  $C_0$  and  $K_H$  to each receiver separately with the following steps:

- The receiver generates a *Channel Setup Request* to the sender with a message containing a 32-bit nonce  $n$ .
- The sender replies with a *First Response* message that contains the hash value of the nonce truncated to 16 bits, together with the correct element  $H_j \in H_C$ .
- At this point, the session key needs to be exchanged. It is worth noting that since it is 80 bits long, a single CAN frame is not enough. Hence, it is split into three parts. The sender forwards three *Consecutive Response* messages, each containing a part of  $K_S$  and  $H_{j+\{1,2,3\}}$ . The receiver can verify the authenticity of each message by checking if  $hash(H_i) = H_{i-1}$ .
- The sender transmits  $C_0$  and  $K_H$ .

All the messages are encrypted with  $K_{SR}$ , except for the last one that uses  $K_S$ . An overview of the process is shown in Figure 6.

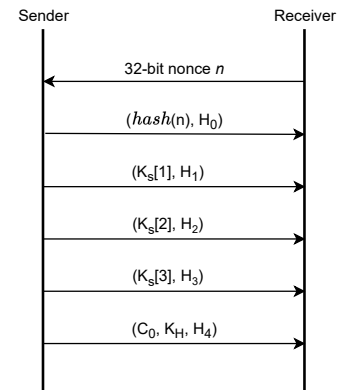


Fig. 6: LCAP Channel Setup performed the first time.

3) *Message Setup*: For every message type the ECU can send, it broadcasts a series of *data messages* containing  $M_0^i \in M_C^i$ . Each message  $i$  is authenticated using the corresponding element  $C_j$ :  $C_1$  can be verified since  $C_0$  was previously shared.

4) *Data Exchange*: True data are exchanged and authenticated by appending the element  $M_j^i \in M_C^i$  and encrypted with the session key  $K_S$ . Each receiver decrypts the message and verifies its authenticity using  $K_H$ : if  $\text{hash}(M_j^i) = M_{j-1}^i$ , the message is accepted.

5) *Chain Refresh*: Since it is periodically necessary to refresh the hash chain  $M_C^i$  before it is completely consumed, the sender must generate a new one in advance. A possible solution may be to compute a new value each time one chain element is used. Then, the ECU provides the new chain to its receivers and authenticates the message with the last hash value of the current chain.

6) *Soft Synchronization*: In case of message losses, the sender and the receiver will be out of synchronization with respect to the hash values of the chain. Hence, the receiver asks for re-synchronization, and the sender replies by delivering the current hash values for each message it sends to that receiver. These are encrypted with  $K_S$  and authenticated as in the *Channel Setup* phase. An overview is shown in Figure 7.

7) *Hard Synchronization*: This phase allows the receiver to ask for the session and HMAC keys in case of message loss during the handshake. This phase develops as in *Soft Synchronization*, with the only difference that before the  $M_j^i$  value, the  $K_S$  and  $K_H$  are exchanged. Authentication is performed as in *Soft Synchronization* as well. An overview is shown in Figure 7.

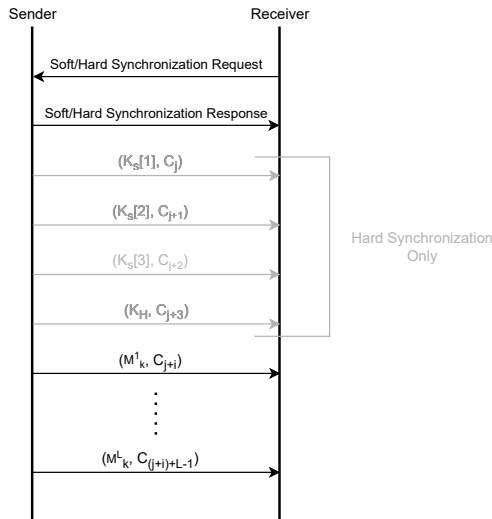


Fig. 7: LCAP Soft/Hard Synchronization. Element  $k$  refers to the current value of the chain.

#### D. LinAuth

LinAuth (2012) is a pair-wise MAC-based authentication protocol [59]. This implies that for each pair of ECUs,  $(ECU_i, ECU_j)$ , there exists a shared secret key  $K_{i,j}$ . Besides,

given a message  $m$  with ID  $k$  ( $m_k$ ), a transmitter ECU computes the MAC values for authentication only for those receiver nodes that will be interested in receiving frames with message ID  $k$ . It follows that (i) to let the receiver understand which is the correct key to use for verification, the authenticated message must also carry the ECU-specific identifier  $ECU_{ID}$  of the transmitting node; (ii) to know which are the nodes for which the transmitter has to compute the MAC value, there is the need to maintain an ID-table that associates the message IDs to the corresponding interested ECUs [59].

When an ECU has to transmit a message  $m_k$ , it computes the corresponding MAC value for each receiver associated with  $ID_k$  listed in the ID-table. Then, it attaches the counter  $C_k$  and the collection  $\{MAC_{k,j}\}$  for all the receivers  $j$ , of the MACs computed. It is worth noticing that, to let the receivers understand which MAC value they need to retrieve for verification, the order in which the MACs are attached must be pre-defined.

To face possible message loss leading to counter out of synchronization counter and to reduce the bus load, authors decide to split the counter into the most and least significant bits:  $C_k = C_k^M || C_k^L$ . Only  $C_k^L$  is transmitted, and the following rules apply for message verification (transmitter  $ECU_i$ , receiver  $ECU_j$ ) [59]:

- If  $C_{k,i}^L > C_{k,j}^L$ , the receiver verifies the MAC value using as counter  $C_k = C_{k,j}^M || C_{k,j}^L$ . If the verification holds, the receiver accepts the message and updates its counter:  $C_{k,j}^L = C_{k,i}^L$ .
- If  $C_{k,i}^L \leq C_{k,j}^L$ , the receivers verifies the MAC value using as counter  $C_k = C_{k,j}^M + 1 || C_{k,i}^L$ . If the verification holds, the receiver accepts the message and updates its counter:  $C_{k,j} = C_{k,j}^M + 1 || C_{k,i}^L$ . If, instead, the verification fails, the message is deemed as replayed and thus discarded.

Lastly, to reduce communication overhead, the authors also designed LinAuth with the possibility of using a group keys approach, thus assigning the ECUs to several [59].

#### E. MaCAN

MaCAN (2012) relies on MACs for establishing trust in the origin of signals on the CAN bus [60]. The primary reason for choosing MACs over asymmetric cryptography is the inherent constraint of the CAN frame's payload size, which makes the latter option impractical. The core components of the MaCAN security framework encompass three key elements: (i) the degree of MAC truncation, (ii) the signing protocol, and (iii) the selection of the freshness element.

1) *MAC Truncation*: To ensure the system's real-time capabilities, MaCAN opts for MAC truncation, limiting the signature length to 32 bits. This truncated signature is included in a single CAN frame alongside the signal. Several factors influence this design choice.

- *CAN Bus Speed* – The relatively low speed of the CAN bus restricts the number of authentication attempts an attacker can make within a given time frame.

- *Session Keys* – Short-lived session keys are employed for ECU communication, limiting the exposure of keys over time.

The typical MAC authentication element length matches the underlying block cipher's block length (e.g., 128 bits with AES). MaCAN, however, considers using lightweight ciphers like PRESENT in real-world implementations [68]. The number of attacker guesses possible determines the truncation level of 32 bits during a session key's lifespan. NIST guidelines usually recommend a minimum 64-bit MAC element, but in MaCAN, due to the slow bus speed and short-lived session keys, 32 bits suffice. This leads to a 1 in 500 risk of guessing a single signature during the key's lifetime. To mitigate frequent attempts, MaCAN suggests monitoring actual vs. expected message frequencies and limiting responses to false signatures.

2) *Signing Protocol*: MaCAN offers two message authentication formats, accommodating multiple signals within most Original Equipment Manufacturer (OEM) defined CAN frames.

- *Dedicated CAN Frame* – This format involves a dedicated CAN frame that carries only one signal and its signature as payload. While this format increases the number of frames needed to transmit the same number of messages, it can be chosen for a limited number of critical signals. However, the limited number of available CAN-IDs and message overhead must be considered.
- *Newly Introduced CAN-ID* – In this format, a new CAN-ID is introduced, and it is included alongside an unauthenticated standard CAN frame. This combined frame simplifies the mapping between signals and signatures, making it suitable for on-demand authentication of typically unauthenticated signals.

3) *Message Freshness*: To safeguard against replay attacks in a unidirectional environment with event-driven signals and potential ECU unavailability, MaCAN employs timestamps for message freshness. A Time Server (TS) is introduced to synchronize time signals across nodes, transmitting timestamps at regular intervals and providing on-demand authenticated timestamps. MaCAN's protocols allow flexibility in defining which signals are signed and the signature frequency relative to message frequency, adapting the security system to the signal's characteristics and resource availability in various scenarios.

4) *Protocols*: MaCAN defines three crucial protocols to enhance CAN bus security: signal authentication, key distribution, and authenticated time distribution.

- *Message Format* – MaCAN introduces a new 6-bit ECU identification token (ECU-ID) for secure communication. A "crypt frame" partitioning scheme is used within the CAN payload to accommodate this. This format allows for direct ECU addressing, consisting of a destination field and crypt message flags. Despite the addition of these security features, the crypt frame format maintains compatibility with standard CAN frames, making it compatible with hardware CAN-ID filters.
- *Signal Authentication* – This protocol governs signal authentication on the CAN bus. The process involves defining the desired signal and its signature frequency in

relation to the message frequency. The protocol employs two message types: request messages (i.e., initial message specifying the requested signal and signature frequency) and authenticated signal messages (i.e., response to the control message).

- *Key Distribution Protocol* – Manages session keys for pairs or groups of ECUs. It introduces a key server (KS) that shares a symmetric long-term key (LTK) with security-enabled nodes on the bus. The protocol's non-time-critical nature allows challenges to ensure message freshness.
- *Authenticated Time Distribution Protocol* – Tackles issues related to non-bidirectional freshness elements like counters or timestamps. A Time Server (TS) provides a reliable, monotonically increasing time signal to all nodes on the bus while safeguarding it cryptographically. Nodes can request authenticated time signals to synchronize their counters or detect tampering.

## F. CaCAN

Centralized authentication in CAN (CaCAN) (2014) follows a centralized authentication approach with the introduction of a central Monitor Node (MN) having the purpose of authenticating all the ECUs of the network and verifying the authenticity of the exchanged messages [61]. Despite the term centralized may also suggest a topology modification of the network, moving from broadcasting to a star topology, this is not the case: the MN is central just from an operational point of view. Besides, the MN is equipped with a special HMAC-CAN controller that can detect and destroy unauthorized frames at runtime by overwriting them with an error frame. To verify messages' authenticity, the protocol requires the MN and ECUs to share a 512-bit cryptographic key, thanks to which the MACs are computed. This key is computed from a pre-shared secret  $S$  assumed to be stored in a Read-Only-Memory (ROM) together with the unique identifier of the ECU.

1) *Authentication and Key Distribution*: The protocol expects a two-way authentication in a challenge-response fashion. Let us consider  $ECU_j$  that needs to authenticate to MN. Assuming that the ECUs can identify each other:

- The MN sends a random nonce  $n$  to  $ECU_j$ .
- As the nonce is received, both the MN and the ECU start to compute the Authentication Code (AC) as

$$AC = \text{SHA-256}(S \parallel n),$$

where SHA-256 is the hash function the authors decided to employ. It returns a 256-bit value.

- After waiting a worst-case computation time for AC, the monitor node sends a data frame to  $ECU_j$ , whose payload comprises a few bits of the AC.
- $ECU_j$  checks for correctness and sends a response with a few continuation bits of the received ones.
- MN checks for correctness.

The AC computed is also used as the cryptographic key for MAC computation. It is worth noting that some key bits are leaked and thus discarded because of the authentication procedure.

2) *Monitoring of Data Frames*: When messages are exchanged, the MN checks if they carry the MAC by checking the message's ID. If so, the monitor node immediately starts computing the HMAC and validating the MAC of the message. If an error is detected, it overwrites the message with an error frame by the end of the transmission. The MAC attached to the message is 1 byte long and derived from keeping just 8 bits of an HMAC algorithm.

$$MAC_i = HMAC(ECU_{ID}, msg_i, FC_i, K_j),$$

where  $ECU_{ID}$  is the unique identifier of the ECU,  $msg_i$  is the payload of the message (without any counter attached),  $FC_i$  is the full counter for message  $i$ ,  $K_j$  is the cryptographic key for sender  $j$ . The authors decided to attach a counter to the message payload to prevent reply attacks. However, due to its 32-bit length (full counter  $FC = UC \parallel LC$ ), it cannot be fully transmitted within the payload. Thus, only the lowest few bits ( $LC$ ) are attached. The MN stores the full counters for each sender and, when receiving a message, checks for counter freshness according to the following policy:

- If  $LC_i = OLC_i$ , the message is discarded as it is classified as a reply attack (where  $OLC_i$  are the holding lowest bits of the monitor node).
- If  $LC_i > OLC_i$ , the message is accepted and  $OC_i$  is updated.
- If  $LC_i < OLC_i$ , the message is accepted,  $OLC_i$  is updated ( $OLC_i = LC_i$ ), and the holding upper value  $OUC_i$  of the counter is increased by 1.

#### G. VeCure

VeCure (2014) is a MAC-based authentication protocol in which ECUs are divided into different *trust groups* according to their trust level [62]. The trust level of an ECU identifies how easily that ECU could be compromised. There is no specification on the number of groups that one can define. Trust groups are hierarchically structured: nodes in the lowest-trust group do not have any secret key, while nodes belonging to the group  $i$  are provided with all the (symmetric) secret keys of groups  $j < i$ , and the secret key  $K_i$  of the group itself. By doing so, nodes in group  $i$  can verify the authenticity of messages transmitted by nodes belonging to groups  $j \leq i$ . Without loss of generality, for VeCure design, authors define two trust groups: high-trust and low-trust [62].

1) *Initialization*: VeCure provides ECU-authentication. Thus, at the initialization phase, each node in the network receives a unique identifier  $ECU_{ID}$  of 1 byte. Besides, ECUs belonging to the high-trust group are provided with: (i) a 128-bit secret key  $K$ , a 2-byte session number  $c_s$ , and message counter  $c_m$ . The session number is a driving session parameter. Thus, ECUs will update it each time the vehicle is started, while the message counter is updated at each message authentication. As  $c_m$  only allows authenticating 65536 messages per session before overflow, ECUs are also required to store an overflow counter  $c_o$  that is incremented each time  $c_m$  overflows (per session). The triple  $(c_s, c_m, c_o)$  uniquely identifies any message that was sent by a specific ECU since the vehicle was initialized [62].

Since each message is bound to the  $ECU_{ID}$ , every node also needs to keep track of the counters of all the other nodes in the same group to verify the authenticity of the received messages.

2) *Authentication Scheme*: Message authentication is achieved via MAC computation over the transmitted data. When a node in the high-trust group has to send some data, it transmits two consecutive messages: the first one is the CAN standard frame and contains the actual data to transmit. In contrast, the second message is structured as  $(ECU_{ID}, c_m, MAC, 0xFF)$ , where  $0xFF$  is a marker used to help the receiver verify that is the follow-up message for authentication, and  $MAC$  is 4-byte MAC value.

Due to the strict requirements for induced delay for authentication, authors design the MAC computation in two phases: a light-weight online calculation and a heavy-weight offline calculation. The MAC value is thus computed as follows:

$$hash = H(ECU_{ID}, c_s, c_o, c_m, K),$$

$$MAC = BME(hash, m),$$

where  $H(\cdot)$  is a one-way cryptographic hash function,  $m$  is the data transmitted, and  $BME$  (Binding, Mapping, Extraction) is a function performing which ensures that there is a one-to-one mapping between  $m$  and the  $MAC$  value [62]. It is worth noticing that the computation of the  $hash$  parameter is independent of the data message  $m$ ; thus, it can be computed offline beforehand for both the sender and receivers. This reduces the induced delay in the communication for MAC computation and verification.

#### H. Woo-Auth

Woo-Auth (2014) is an HMAC and session keys-based authentication protocol [12]. The gateway ECU (GW) is assumed to have a higher computing power than standard ECUs. Each ECU  $i$  and the GW are provided with their own unique identifier  $ECU_{ID_i}$  and  $GW_{ID}$ .

1) *Loading Long-Term Symmetric Keys*: Each  $ECU_i$  loads in a secure storage and through a secure channel, its long-term symmetric key  $K_i$  and  $K_{GW}$ , while the GW loads all the keys  $\{K_i\}$  for each node  $i$  and its own key  $K_{GW}$ . Key  $K_i$  is used between the GW and  $ECU_i$ ,  $K_{GW}$  instead is shared among GW and all the ECUs.

2) *Distribution of Initial Session Keys*: When starting the vehicle, each ECU begins a session key derivation process with the GW in a predefined order according to the following steps.

- $ECU_i$  selects a random value  $R$  and sends it to GW.
- GW selects a random number  $S$  and generates the MAC:

$$MAC_1 = H1_{K_i}(ECU_{ID_i}, GW_{ID}, R, S),$$

where  $H1$  is the keyed hash function using  $K_i$  as key and giving a 64-bit output. Then, it sends  $MAC_1$  to  $ECU_i$  together with  $S$ .

- $ECU_i$  verifies  $MAC_1$  and computes the initial session keys.

$$K_E \parallel K_A \parallel K_{EK} \parallel K_G \parallel K_U = KDF(K_{GW}, S),$$

where  $KDF(K_{GW}, S)$  is the keyed one-way hash function using key  $K_{GW}$  and  $S$  as parameters;  $K_E$  is the encryption session key for CAN data frames;  $K_A$  is the authentication session key for CAN data frames and session key update;  $K_{EK}$  is the encryption key for the session key update phase;  $K_G$  is the key used for the following session key derivation process;  $K_U$  is a key used for external devices communication.

- $ECU_i$  generates the following:

$$MAC_2 = H2_{K_i}(ECU_{ID_i}, S),$$

$$MAC_3 = H1_{K_A}(ECU_{ID_i} \parallel K_{GW}),$$

where  $H2$  is a keyed hash function using key  $K_i$  and returning a 32-bit output. Then, it sends  $MAC_2, MAC_3$  to GW.

- GW verifies  $MAC_2$  to check if  $S$  was correctly received. Afterward, it starts the session keys derivation as done by  $ECU_i$ .
- With the session keys, GW can now verify  $MAC_3$  to be sure they share the same keys.

3) Authentication of CAN Frames: Due to the restricted payload size of CAN frames, authors propose two possible solutions for sending messages: a *basic method* in which the output of the encryption is truncated to fit the payload size, and an *enhanced method* in which the encryption result is split into two parts and sent through two subsequent messages. For both approaches, the encryption and decryption procedures are the same. The authors propose to use AES-128 for encryption and Keyed-Hash MAC for authentication.

Let us assume to be at the  $k^{th}$  session. Sender  $ECU_i$  generates the ciphertext  $C$  and the authentication tag  $MAC_i$  as follows:

$$C = \text{AES-128}_{K_{E,k}}(CTR_{ECU_i}) \oplus M,$$

$$MAC_i = H2_{K_{A,k}}(ECU_{ID_i} \parallel C \parallel CTR_{ECU_i}),$$

where  $M$  is the message payload, and  $CTR_{ECU_i}$  is the counter of  $ECU_i$  used for freshness. The MAC is split and inserted into the first 16 bits of the IDE field (the two remaining unused bits are set to zero) and the remaining part in the CRC field. Notice that, as the CRC field contains part of the MAC, which aims to provide integrity and authentication, the CRC field still can be considered as proof that the message has not been altered. The receiving ECU first verifies the MAC received and then decrypts the message as follows.

$$M = \text{AES-128}_{K_{E,k}}(CTR_{ECU_i}) \oplus C.$$

Lastly, it increments the counter associated with the sender ECU.

4) Session Key Update: Encryption and authentication keys are periodically updated according to the following mechanism.

- 1) The GW selects a new random value  $S_{k+1}$  and broadcasts a *Key Request Message* whose payload is given by  $(C \parallel MAC)$ , where

$$C = \text{AES-128}_{K_{EK,k}}(CTR_{GW}) \oplus S_{k+1},$$

$$MAC = H_{K_{A,k}}(GW_{ID} \parallel C \parallel CTR_{GW}).$$

- 2) Every ECU, receiving the *Key Request Message*, verifies the message and derives the session keys to be used in the  $(k+1)^{th}$  session  $K_{E,k+1} \parallel K_{A,k+1} \parallel K_{EK,k+1} \parallel K_{G,k+1} = KDF_{K_{G,k}}(S_{k+1})$ . Also, frame counters are initialized to zero. It is worth noting that to derive the new session keys for  $k \geq 2$ , we use the key  $K_{G,k-1}$ .
- 3) Each ECU generates a *Key Response Message* with payload

$$M = H1_{K_{A,k+1}}(ECU_{ID_i} \parallel S_{k+1}),$$

and transmits it to GW to confirm the reception of the previous request.

- 4) After receiving the confirmation response, the gateway sets the counters for the corresponding ECUs to zero. Then, when all response messages are received, it can set its counter to zero.

### I. LeiA

Lightweight Authentication Protocol for CAN (LeiA) (2016) makes use of session keys and lightweight cryptographic primitives to assure a good level of security [7]. Moreover, LeiA is the first protocol compliant with AUTOSAR specifications (release 4.2) [50]. Each ECU needs to store a tuple  $(ID_i, K_i, e_i, K_{S,i}, c_i)$  for each message type  $i$ , and the security parameter  $\eta$ , where:

- $ID_i$  is the CAN ID for data type  $i$ ;
- $K_i$  is a 128-bit long-term symmetric key used to generate the session key;
- $e_i$  is a 56-bit epoch value incremented at each vehicle start-up or when the counter  $c_i$  overflows;
- $K_{S,i}$  is a 128-bit session key used to generate the MACs;
- $c_i$  is a 16-bit counter included in the MAC computation.

1) Setup: ECUs are initialized by generating a tuple  $(s, n_s)$  from the security parameter  $\eta$ . The parameter

$$s = \langle K_0, \dots, K_{n-1} \rangle$$

is a collection of master keys, one for each ID, while parameter

$$n_s = \langle (c_0, e_0), \dots, (c_{n-1}, e_{n-1}) \rangle$$

is public and is a collection of epoch and counter values couples. Both counters and epochs are initialized to zero.

2) Session Key Generation: For each  $ID_i$  the *Key Generation Algorithm* (KGA) is used to derive the corresponding session keys according to the following procedure:

- $e_i = e_i + 1$ ;
- $K_{S,i} = KGA(K_i, e_i)$ ;
- $c_i = 0$ .

3) Sending Authenticated Messages: After the generation of the session keys, the ECU can authenticate messages. It must first update counter  $c_i$  and generate the following authentication tag.

$$MAC_i = AGA(K_{S,i}, c_i, msg),$$

where  $AGA$  is the *Authentication Generation Algorithm* and  $msg$  is the payload of the message. If  $c_i$  overflows,  $e_i$  is

incremented to compute a new session key. Afterward, the sender will transmit  $c_i$ ,  $msg$   $MAC_i$ . On the other hand, after reading and checking for counter freshness, the receiver updates it and verifies the MAC. For any message, the counter is placed in the extended identifier field preceded by a 2-bit command code that specifies the content of the payload according to the encoding:

- 00 for normal data;
- 01 for the MAC of the data;
- 10 for the epoch value;
- 11 for the MAC of the epoch.

From this encoding, we can understand that the MAC is sent in a subsequent message with respect to the one containing  $msg$ . Moreover, MACs are transmitted with a different ID with respect to the message they are authenticating. The authors propose to set  $ID_{MAC} = ID_{data} + 1$ .

4) *Resynchronization*: The resynchronization procedure is used when a MAC cannot be verified, and the receiver sends an error signal. In this case, the sender broadcasts a message containing its current  $e_i$ ,  $c_i$ , and the authentication tag  $MAC(e_i)$  for the epoch value. This allows receivers to resynchronize their epoch and counter. It is worth noting that receivers will update the two parameters only if the received values are higher than the stored ones. Indeed, if the counter's value were lower than the receiver's, it would be a sign of a replay attack. In this case, the ECU must not update the stored counter.

#### J. vatiCAN

Vetted Authenticated CAN (vatiCAN) (2016) is based on HMAC authentication, and to reduce bandwidth overhead, only safety critical messages are authenticated [63]. The choice of which IDs to protect is manually performed during development. Each ECU is provided with a table with IDs expected to be authenticated and 128-bit cryptographic keys for each message type it can transmit among them. Besides, the protocol also allows grouping related IDs together to use a unique key. Of course, the key for each ECU or group needs to be provisioned to any other ECU that will receive the corresponding messages. These keys are assumed to be stored in ECUs at manufacturing time. A 64-bit MAC can authenticate exchanged messages returned from an HMAC function (authors suggest using the SHA-3 function). The MAC for  $ID_i$ , or group  $\mathcal{G}_j$ , is computed as follows.

$$MAC_i = HMAC(ID_i \parallel msg \parallel c_i),$$

where  $ID_i$  is the CAN identifier of the message type  $i$ ,  $msg$  is the payload and  $c_i$  is an ID-specific counter. The ID of the message is considered for MAC computation to prevent the same payloads with different identifiers from having the same MAC if they share the same key. The MAC is sent in a subsequent frame, which is done with a different ID concerning one of the authenticated messages. To maintain the priority relationship between messages, the choice is to pose  $ID_{MAC} = ID_{message} + 1$ . Because of possible losses, an ECU may remain out of synchronization, i.e., not aligned with the counter  $c_i$ . This situation will lead the considered

receiver to reject the messages related to  $ID_i$  or  $\mathcal{G}_j$  since the MACs it computes will not correspond to those received. For this reason, authors introduced a *Nonce Generator* (NG) node. The NG periodically broadcasts a random global nonce  $g$  to be used as the new starting value for all counters. In other words, the NG node resets the counters, giving a new starting point. The authors suggest a generation frequency of the new counter of 50 ms.

#### K. LiBrA-CAN

Lightweight Broadcast Authentication for CAN (LiBrA-CAN) (2017) is designed to protect against internal attackers, and it is based on two paradigms: *key splitting* and *MAC mixing* [64]. The protocol is designed to work upon both CAN-FD and CAN+ and in doing so, it benefits from their features, achieving better performance compared with the standard CAN. The following assumptions hold.

- The adversary can be an external device plugged into the network or a legitimate ECU that has been compromised. In both cases, the corrupted nodes are assumed to be in the minority.
- Cryptographic keys are assumed to be stored in secure memory and distributed to the nodes through a secure key distribution mechanism. Keys are renewed at each protocol initialization and refreshed at periodic intervals. There is no discussion on the specific key generation and sharing algorithms.
- Since the protocol uses counters, a resynchronization mechanism to align nodes to the correct counter value is assumed to be in place.
- ECUs  $i$  are provided with unique identifiers  $N_i$ .

The authors also propose two other approaches based on master and distributed-oriented authentication. However, for our work, we present only the main proposed scheme.

1) *Mixed Message Authentication Codes*: The Mixed Message Authentication Codes (M-MACs) technique is at the basis of the protocol and consists of aggregating multiple MACs into a single one. The M-MAC uses an array of keys to build a tag that is verifiable by any of these keys. The M-MAC mechanism is required to satisfy two fundamental security properties: *unforgeability*, which is standard for MACs and ensures that an adversary is not able to forge an authentication tag correctly, and *strong non-malleability*, which allows a verifier to detect whenever an adversary had tampered with any part of the M-MAC. A mixed message authentication code is a tuple  $(Gen, Tag, Ver)$  of probabilistic polynomial-time algorithms such that:

- $\bar{K} \leftarrow Gen(1^l, s)$  is the key generation algorithm. It takes in input a security parameter  $l$  and outputs a key set  $\bar{K} = \{K_1, \dots, K_s\}$  of  $s$  keys.
- $\tau \leftarrow Tag(\bar{K}, M)$  is the MAC generation algorithm. It takes as input the key set  $\bar{K}$  and the message tuple  $M = (m_1, \dots, m_s)$ , and outputs a tag  $\tau$ .
- $v \leftarrow Ver(k_i, m_i, \tau)$  is the verification algorithm. It takes in input the key  $k_i \in K$ , the message  $m_i$ , the tag  $\tau$  and outputs a single bit  $v$ , equal to 1 if and only if the tag is valid with respect to the used key and input message.

The easiest way to build the M-MAC is by concatenating multiple tags.

2) *LiBrA-CAN Main Scheme*: Given  $n$  nodes placed in groups of size  $g$ , the protocol develops into the following steps.

- **Setup** – This is the key setup phase that generates  $t$  random  $l$ -bit keys. The parameter  $t$  is given as  $t = \binom{n}{g}$ , representing the number of subsets of size  $g$  out of  $n$  nodes. Each node is then provided with the keys for its group. Let us define  $\bar{K}^i = \{K_1^i, \dots, K_{t'}^i\}$  the set of keys for node  $N_i$ , where  $t' = \binom{n-1}{g-1}$ , and  $\bar{K}^{i,j} = \{K_1^i, \dots, K_{t''}^i\}$  be the set of share keys of node  $N_i$  with  $N_j$ , where  $t'' = \binom{n-2}{g-2}$ .
- **Send authenticated message** – When node  $N_i$  wants to broadcast message  $m$ , it increments its local counter and computes the corresponding M-MAC using its key set  $\bar{K}^i$ . Then, the message is sent together with its M-MAC. It is worth noting that in this case, the M-MAC construction receives as input  $\bar{M} = (m, m, \dots, m)$  composed by  $s$  repetition of the same message  $m$ .
- **Verification** – Node  $N_j$  receives an authenticated message sent by node  $N_i$ . Then, it first checks for freshness, and then if  $v = 1$  for all the keys belonging to  $\bar{K}^{i,j}$ , the message is then assumed to be authentic, and the counter is updated.

#### L. VulCAN

VulCAN (2017) employs MACs to provide strong message authentication for secure communication. The paper articulates an attacker model where malicious parties have remote access to the car's internal network and can thus perform arbitrary message manipulation and code execution. Based on the problem statement, authors identify four requirements closely following previous research: (i) message authentication, (ii) lightweight cryptography, (iii) replay attack resistance, and (iv) backward compatibility. On top of these requirements, they provide five guarantees that authors claim were not achieved by state-of-the-art authentication schemes: (i) real-time compliance, (ii) component isolation, (iii) component attestation, (iv) dynamic key update, and (v) secure legacy ECU integration.

1) *MAC Generation*: VulCAN associates a symmetric 128-bit cryptographic key with each authenticated CAN identifier for message authentication. This key generates a 64-bit MAC over the message identifier (ID) and payload, including a monotonically increasing counter to protect against replay attacks. The MAC generation process is as follows.

- For a message with identifier  $i$ , payload  $p$ , and counter  $c_i$ , the MAC is computed as

$$m = MAC(key_i, (i|p|c_i)).$$

- This MAC is transmitted as the payload of a separate CAN message with the ID  $i + 1$ .

This approach avoids priority inversion issues in the CAN bus, as CAN identifiers also serve as priority indicators during bus arbitration. In cases where the identifier  $i + 1$  is already in use by a legacy application, a different application-specific authentication ID is selected to maintain compatibility with legacy ECUs.

2) *Nonce Initialization and Resynchronization*: VulCAN includes nonce values in the MAC computation to address replay attacks and packet loss. Nonces ensure the same counter values are not reused under the same cryptographic key and associated data. Nonce initialization is typically handled with short-term session keys generated at system boot time or when the session counter overflows. Nonce resynchronization mechanisms are implemented to deal with packet loss scenarios, ensuring that sender and receiver nonces remain synchronized. Depending on the application, unused parts of the message payload may be used to encode nonce values.

#### M. TOUCAN

TOUCAN (2019) makes use of the Chaskey MAC algorithm and AES-128 encryption to authenticate and secure CAN communications, and it is designed to be AUTOSAR compliant (release 4.3.1) [66]. Authors assume that all the needed cryptographic material is already available to each ECU, which then shares 128-bit cryptographic keys for authentication and encryption. The authentication tag is computed using the Chaskey MAC algorithm, an efficient permutation-based MAC algorithm [69]. Chaskey receives in input a 128-bit key and a message split into 128-bit blocks, to which it applies a permutation  $\pi$ . If the last block is incomplete, padding values are inserted to reach the desired length. The permutation  $\pi$  is applied eight times for each block  $m_i$  performing an Addition-Rotation-XOR (ARX) permutation. This means that a sequence of three operations is applied for each block: addition mod  $2^{32}$ , bit rotation, and XOR operation. The output of the Chaskey algorithm is a tag  $\tau$  of  $t < n$  bits, where  $n$  is the length of the key. The authors propose to dedicate 40 bits out of the 64 available for the payload and the remaining 24 bits for the authentication tag.

#### N. AuthentiCAN

AuthentiCAN (2020) is a protocol built on CAN-FD and implements encryption and authentication by using asymmetric cryptographic primitives [67]. This choice is motivated by the authors wanting to avoid using a single shared key since this could be a serious issue in case of a compromised ECU. ECUs are assumed to be provided with hardcoded key pairs and their unique identifiers  $ECU_{ID}$  in a secure memory at the manufacturing phase. The protocol develops through four different *Transmission States* discriminated by the first two Most-Significant-Bits (MSB) in the frame's payload.

1) *Broadcast Public Key (00)*: The payload is the transmitter's public key sent in clear so any node can read it. This phase is performed in a predefined order. The choice of broadcasting the public key rather than hardcoding the other nodes' public keys on the ECU is because faulty nodes can be detected and replaced, or other new nodes can be added without the need to hardcode the new keys.

2) *Send Nonce List (01)*: A node entering this state wants to send a nonce list after exchanging the public keys or because a previous list is exhausted. In the first case, the ECU generates a list of all the other nodes and encrypts it with the

respective public key. In the second case, only the new list for a transmission with a specific node is generated and delivered. It is worth noting that node  $A$ , to communicate with a node  $B$ , uses the nonces in the list sent from  $B$ , and vice-versa. Of course, they can send only as many messages as many nonces in their lists, and when the list is exhausted,  $B$  needs to request  $A$  for a new list. The authors decided to set the size of the nonces to 8 bits and to fill the list with as many nonces as possible. Moreover, they also decided to design the protocol to avoid making the nodes send the nonce list at the startup but rather just when actually needed for communication.

3) *Send Message (10)*: A node enters this state when it wants to communicate with another node by sending actual data. In this case, the payload contains the encrypted message consisting of the concatenation between the plaintext and the correct nonce. After message delivery, the sender erases the nonce used. Thanks to encryption with the receiver's public key, only the legitimate receiver can decrypt it and check if the nonce matches what is expected.

4) *Synchronize Nonce List (11)*: A node enters this state when it has no more nonces available for communication with another node or after a fixed number of rejected messages due to a possible message loss, which makes the two ECUs misaligned with respect to the counter list. The payload contains the receiver's  $ECU_{ID}$  in plaintext. The receiver is then triggered to enter in the *Send Nonce List* state. As it may also happen that a node sends a resynchronization request before the actual nonce list is exhausted, the receiver locally deletes its internal list and generates a new one.

## O. S2-CAN

Unlike all the others presented so far, S2-CAN (2021) is an authentication protocol that aims to offer authenticity, confidentiality, and freshness without using cryptography [39]. Indeed, the security of the protocol depends on (i) randomly generated internal IDs and counters to offer authenticity and freshness, (ii) frames payload cyclic shifting of a random integer to offer secrecy and confidentiality. Despite the protocol not using cryptography for message authentication, it is a session-based protocol. Hence, some cryptographic primitives for securing the session parameters are still needed. To this purpose, ECUs are equipped with pre-installed symmetric keys. Furthermore, the protocol also requires a logical ordering among ECUs. The session parameters consist of the following.

- Global 3-byte encoding parameter  $f$ .
- ECU-specific integrity parameter  $int\_ID_j$ . It is a 1-byte randomly generated internal identifier of the  $j$ -th ECU ( $ECU_j$ ).
- ECU-specific integrity parameter  $pos_{int,j}$  for  $ECU_j$ . It specifies the random position within the CAN payload where the internal identifier will be located. This parameter is needed since the ECU internal ID is embedded inside the free space of the CAN payload, which can change each time. Authors studied in detail how much free space is available on average [39].
- ECU-specific 2-byte counter value  $cnt_j$  for  $ECU_j$ . It is randomly generated and represents the starting value for the counters used to avoid replay attacks.

1) *Handshake*: The handshake phase, depicted in Figure 8, develops through three stages for each new session  $S_i$ , where  $i$  is the number of the session, and repeats with a fixed periodic interval of period  $T$ . The gateway ECU, also named Master ECU ( $ECU_M$ ), handles the handshake phase and establishes new sessions.

- **Stage 1 - Initialization.**  $ECU_M$  sends an initialization message  $msg_{init}$ , identified by a specific standard CAN identifier, to indicate the beginning of the handshaking phase for establishing a new session  $S_i$ . The payload of this message includes a randomly generated encoding parameter  $f_i = (r_0, r_1, \dots, r_8)$ , where  $r_l \in [0, 7]$  represents the bit rotation number for the  $l^{th}$  byte in the 8-byte CAN payload. Each  $r_l$  can be represented with 3 bits. Moreover, a 2-byte counter  $cnt_0$  (not to be confused with the ECU-specific session parameter  $cnt_j$  previously defined) is also attached for replay attack defense. The message is encrypted with the pre-shared symmetric key  $K$  and authenticated thanks to a 32-byte SHA256-HMAC of the previous 5 bytes. It is worth noting that since only 3 bytes are left, the MAC should be truncated. However, as authors claim that a 3-byte MAC is too short, they decided to split  $msg_{init}$  into two consecutive messages  $msg_{init,1}$  and  $msg_{init,2}$ . These messages contain payload  $p_1$  and  $p_2$ , respectively. Payload  $p_2$  is composed then by the following 8 bytes of the MAC. This results in a total of 11 bytes MAC. Finally, AES-128 is used for encryption.
- **Stage 2 - Acknowledgment.** ECUs decrypt  $p_1$  and  $p_2$  and extract the encoding parameter  $f_i$ . Afterward, following the established ordering, each  $ECU_j$  sends back an acknowledgment message  $msg_{j,ACK}$  containing a 1-byte positive acknowledgment code  $ack$  and the three ECU-specific parameters ( $int\_ID_j$ ,  $pos_{int,j}$ ,  $cnt_j$ ). Besides, to provide integrity and freshness protection, the acknowledgment message must include a 2-byte handshake counter  $cnt_i$  and the truncated HMAC of the message. As in *Stage 1*, two messages  $msg_{j,ACK,0}$ ,  $msg_{j,ACK,1}$  are needed. It is worth noting that, to avoid internal identifier collisions, an ECU must discard those IDs that were generated by the previous ECUs.
- **Stage 3 - Finalization.**  $ECU_M$  completes the handshake phase by sending a final message  $msg_{fin}$  to signal it has successfully received all the acknowledgment messages from the ECUs. A specific CAN ID identifies this message, and its payload comprises a random non-zero payload (again split into two messages).

If any acknowledgment message delay exceeds a certain threshold, the handshake times out, and  $ECU_M$  will restart it from *Stage 1*. If the handshake is still unsuccessful after several attempts, all the ECUs can revert to the standard CAN communication until the next start of the vehicle.

2) *Operation*: This consists of the regular exchange of data. To save space in the payload, a 2-byte parameter  $q_j$  is computed from the previously stored parameters.

$$q_j = LEFT\_ZERO\_PAD(int\_ID_j, 8) \oplus cnt_j.$$



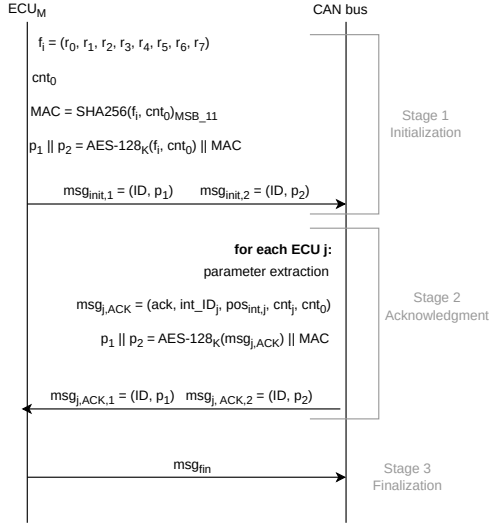


Fig. 8: S2-CAN handshake process.

The message's payload is first logically XORed with  $q_j$ , and then a Circular Shift (CS) operation is applied according to the previously stored encoding parameter  $f_i$ . This CS operation is a byte-wise bit rotation to the  $l^{th}$  byte according to the value of the corresponding  $l^{th}$  element of  $f_i$ . The encoded message is then broadcast in the bus with the ECU-specific counter  $cnt_j$ , which will be incremented for the next message. The receiving ECUs will reverse the encoding process and check for the message's authenticity, integrity, and freshness. Possible losses may lead to an out-of-synchronization scenario. For this reason, receivers can accept packets with a counter value greater than expected within a certain threshold, depending on the packet loss rate.

## VII. COMPARISON CRITERIA

This section outlines the comparison criteria we identify to highlight the strengths and weaknesses of the considered protocols. We distinguish between two classes of criteria: *security criteria* and *operational criteria*. The security criteria identify those properties necessary for a secure and effective authentication protocol for CAN. The operational criteria are features, apart from the security capabilities, that one may consider when deciding which authentication protocol to adopt. These operational features do not change the ability of the system to be resistant or not to some vulnerability. Rather, they can help enhance the security and reliability of the system. For example, using session keys does not make a system resistant to replay attacks, but it can harden for the attacker to succeed in the attack [70]. This way, the operational criteria integrate into the security requirements, offering a complete view of the protocols.

Section VII-A outlines the identified limitations of the defined requirements in the related works. Section VII-B and Section VII-C present the identified security requirements and operational criteria, respectively, upon which we carry out our comparative analysis of the protocols. A summary of the comparison criteria is presented in Table IV.

### A. Related Works Limitations

Nowdehi *et al.* [30] define five industrial requirements that compare the considered protocols: effectiveness, backward compatibility, support for vehicle repair and maintenance, sufficient implementation details, and acceptable overhead. However, we point out the following limitations regarding the definition of these requirements.

- *Cost effectiveness* – According to Nowdehi *et al.* [30] definition: “[...] requiring all ECUs to have hardware-supported cryptographic primitives to achieve necessary performances for safety-critical real-time systems is not cost-effective”. In the definition, the authors are mostly referring to HSMs, since from the analysis of the protocols given in Section VI, these are the hardware modules that are used to store the cryptographic primitives. While on the one hand, we acknowledge that their requirements increase the cost of production, on the other, it is important to point out that without those modules, any authentication protocol based on secret keys would be completely insecure. Indeed, probing attacks can be carried out in these unprotected scenarios, and the loss of the key confidentiality would make any authentication attempt useless [71], [72]. Although the cost of an ECU equipped with an HSM might vary depending on its computational capabilities [73], some works in the literature are focused on the minimization of the number of those modules in automotive applications [74], [75], [76], [77]. Furthermore, as detailed in Section IV, different types of HSMs exist based on their security level, granting additional flexibility during the design process of the vehicle network.
- *Backward compatibility* – According to Nowdehi *et al.* [30] definition: “[...] backward compatibility means that a new solution must be able to co-exist with existing technologies and implementations. [...] Consequently, only a message authentication solution that manages the authentication in an extra frame can be considered backward compatible”. Limitations of such a definition of backward compatibility are (i) the fact that nodes that are not required to provide authentication can be firmware updated in such a way they ignore the authentication part, and (ii) providing authentication in a subsequent frame may raise delay issues, as we will point out better later in our analysis. Besides, the authors did not consider the trend of shifting to the CAN-FD protocol (which is considered backward compatible) as the new standard, which would allow for a much larger payload transmission [41], [78]. Hence, the authentication bits can be inserted into the additional space in the payload, and those ECUs that do not deal with authentication can ignore those bits.

Lastly, we highlight that the requirements defined by Nowdehi *et al.* [30] are implementation and performance-oriented. At the same time, they can not provide an understanding of the security level and possible vulnerabilities of the considered protocols. Therefore, no security analysis is carried out, meaning the security criteria and possible vulnerabilities

TABLE IV: Security and operational criteria definition for the protocols' comparative analysis.

Type	Property	Description
Security Criteria	Atomic authentication	Authentication data is transmitted together with authenticated message
	Replay attack resistance	Nodes can discriminate the freshness of the received messages and discard already transmitted ones
	Masquerade attack resistance	The receiver can discriminate whether a legitimate or rogue ECU sends the message
	Hard real-time compliance	Employed cryptographic algorithms must be considered as fast algorithms
	Resynchronization capability	Ability of the system to restore the alignment with respect to the parameters between devices in case of message loss
Operational Criteria	Backward compatibility	No hardware or topology modifications applied to the already existing one
	Group keys	A same key is used to authenticate correlated message IDs. In particular: <ul style="list-style-type: none"> <li>• Use of one key for each message ID, or</li> <li>• ECUs that can send any <math>ID \in G_i</math> can also send all the other IDs belonging to that group</li> </ul>
	Session keys	Periodic refresh of the authentication keys
	Additional storage memory	Additional amount of memory needed for the handling of the authentication parameters and keys

of the proposed authentication mechanisms are not evaluated. On the contrary, providing a security analysis of authentication protocols for CAN is the major purpose of this work. To the best of our knowledge, ours is the first work of this kind.

### B. Security Criteria

In the following, we define the security requirements an authentication scheme for CAN shall comply with for in-vehicle communications.

- **Atomic Authentication** – The authentication process must be an atomic operation [79]. This means that any extra data needed for the authentication process cannot be separated from the message it belongs to. Non-compliance to this requirement would (i) increase the bus load and (ii) delay the overall process, possibly causing serious issues as the receiving node could not process the received data [80], [81], [82].
- **Replay Attack Resistance** – This property ensures that the system can discriminate the freshness of a received message, thus correctly deciding whether to accept or discard it. The verifying nodes should discard messages intended to replay previous authenticated messages. The replay attack is one of the most common, effective, and threatening attacks that can be easily performed on the standard CAN bus [83]. It can be performed even in a trustworthy environment and may affect the vehicle's security in critical scenarios [84].
- **Masquerade Attack Resistance** – It is an attack in which a malicious entity pretends to communicate with the victim by impersonating another legitimate entity [85]. Specifically for the CAN framework, a malicious ECU impersonates another one by transmitting frames with IDs it is not expected to transmit. The rogue ECU can either be an external one (external masquerade attack) [86] or a compromised legitimate one (internal masquerade attack) [87], [88], as discussed in Section V. Since CAN IDs do not identify the transmitter but the message's priority, an authentication protocol shall allow the receiver to discriminate whether it is a legitimate or rogue ECU

sending the message. Within the internal masquerade attacks, we distinguish between two types of attacks:

- *General-internal*: this is the general case of an internal masquerade attack in which a rogue ECU attempts to transmit and authenticate messages with IDs that the ECU was not supposed to transmit. Particular attention should be given when using the group keys approach. Indeed, it may be the case in which an ECU belongs to a certain group  $G$  but for which it can transmit only a subset of the IDs of that group. Suppose the ECU is compromised. In that case, the membership to the group  $G$  allows the attacker to know the cryptographic key also for the IDs the ECU was not supposed to use, but which it can correctly authenticate anyway as it has the correct secret key and related parameters.
- *Selective-internal*: this is a special case in which symmetric keys are employed, and an ECU is interested in receiving message(s) with certain ID(s), but that it can not transmit messages with those ID(s). Although the ECU does not transmit those types of messages, to verify their authenticity at reception, the ECU must be provided with the corresponding secret keys anyway, thus opening possible breaches for an internal masquerade injection. Indeed, by having the key, the ECU may be able to authenticate forged messages correctly.

During our analysis, we will refer to *Internal Masquerade Attack* when no distinction between *General-internal* and *Selective-internal* is made, thus considering both scenarios.

- **Hard Real-Time Compliance** – As on-board networks handle safety-critical messages whose arrival time is a critical aspect, the hard real-time constraint is a must [89]. To meet this requirement, the cryptographic mechanisms used for the authentication shall be fast and do not add any large overhead [90]. We make the following assumptions.
  - 1) Comparison, counter increment, and message build operations are fast with negligible execution time. Therefore, we only focus on cryptography-related operations.

- 2) Protocol setup or session parameter exchange phases are amortized in time as they are periodically performed and not at every message authentication. Thus, we focus only on the authentication transmission.
  - 3) HMAC algorithms are fast. This assumption is sustained by the results presented in *Groza et al.* [32], in which the authors report that encryption algorithms like AES-128 or SHA-256 are suitable for practical applications.
- **Resynchronization Capability** – The system can restore the alignment with respect to the parameters between devices in case of message loss [91]. Indeed, losses may lead ECUs to be out-of-synchronization, causing the impossibility of successfully authenticating any message, which in turn makes ECUs continuously discard them. Hence, a proper resynchronization mechanism is necessary for system reliability.

### C. Operational Criteria

In the following, we define operational features we consider for a complete evaluation of a protocol for CAN authentication.

- **Backward Compatibility** – This is one of the biggest problems encountered while developing an authentication protocol for CAN [92]. From our perspective, one of the most important aspects is that to be fully backward compatible with the CAN standard protocol, no hardware or topology modifications should be applied. We exclude from our definition of hardware modification the inclusion of possible HSMs, as we believe these modules are essential to provide authentication in in-vehicle communications, and future ECUs shall be provided with HSMs by default. While other surveys discussed in Section II also consider software updates on ECUs to invalidate this compatibility, we argue that the ability of a system to introduce authentication just by updating its firmware makes it backward compatible since it does not require physical modification [93]. Furthermore, some surveys also consider the addition of specific frames in the payload or other fields of the CAN message to invalidate the backward compatibility since other ECUs might not be able to interpret them correctly [30]. However, since some fields of the CAN messages are left unused for this exact purpose, ECU that are not involved in the authentication process ignore them by default. Moreover, given that ECUs use specific encodings to exchange messages, the payload often has free bits that are not utilized, providing free space for the exchange of the MAC values [94].
- **Group Keys** – It is reasonable to think that if the same key is used to authenticate correlated message IDs, then the size of key storage will reduce. In general, ensuring this requirement is not very demanding [13]. However, this solution must be carefully designed and weighted against the advantages it could bring. As anticipated, we stress that using the same key for authenticating more than one ID provides opportunities for masquerade attacks: a compromised ECU could send a message with

an ID that was not expected to but that belongs to the same group of another ID that is allowed to send [95], [96]. Hence, that ECU can authenticate the message correctly as it possesses the correct key. This would make the attack impossible to detect unless with any additional sophisticated control mechanism [97]. Nonetheless, additional IDSs might also introduce new attack vectors to exploit [98]. The worst case occurs when using just one single key. Because of these motivations, we decide to redefine the group keys feature as either (i) the use of one key for each message ID, (ii) or as the case in which the ECUs that can send any  $ID \in G_i$  can also send all the other IDs belonging to that group.

- **Session Keys** – To defend against replay attacks, the most common solution is introducing a counter value somewhere during the authentication or encryption process to ensure the freshness of the incoming message. To make this solution effective, the counter shall not be reused. Note that the counter may overflow its maximum value if there are many authenticated messages. A common solution is to adopt a session key approach that allows the reuse of the same counter value when the encryption key changes. Also, this limits the damages due to key disclosures for the key's lifetime, after which the new key must be learned.
- **Storage Overhead** – The storage memory available is an important aspect to consider when designing an in-vehicle system [99]. The available storage for an ECU impacts the authentication protocol's security level since it influences several security parameters, such as the length of the nonces and encryption keys. Besides, the amount of available storage per ECU also influences the overall system's implementation cost, which is another important aspect to consider. In our analysis, we consider only the additional storage, with respect to the standard CAN, required by the ECUs to implement the authentication protocol. We do not consider temporary values like intermediate parameters or authentication tags/signatures, as they are of interest and stored just for the time needed, after which the space can be easily freed.

## VIII. EVALUATION OF SECURITY CRITERIA

In this section, we provide a comparison of the security criteria offered by the presented protocols in light of the security requirements defined in the previous section, which are atomic authentication (Section VIII-A), replay attack resistance (Section VIII-B), masquerade attack resistance (Section VIII-C), hard real-time compliance (Section VIII-D), and resynchronization capability (Section VIII-E). Table V summarizes results from our analysis.

### A. Atomic Authentication

**CANAuth** does not properly send the authentication tag with the authenticated message. Still, since it is sent through the overclocked bits of CAN+, they can be considered unique messages. Indeed, no other ECU can interpose between the

TABLE V: Security criteria comparison of the different protocols.

Protocol	Atomic Authentication	Replay Attack Resistance	Masquerade Attack Resistance			Hard Real-Time Compliance	Resynchronization Capability
			Gen. Int.	Sel. Int.	Ext.		
CANAuth [13]	✓	✓	✓**	✗	✓	✓	✓
Car2X [58]	✗	✗	✓	✓	✓	✗***	Not needed****
LCAP [14]	✓	✓	$(2^{16} - n)^{-1}$ -secure			✓	✓
LinAuth [59]	✓	✓	✓	✓	✓	With group keys: ✓ Without group keys: ✗	✓
MaCAN [60]	✓	If not using group keys	$500^{-1}$ -secure	✗	✓	✓	✓
CaCAN [61]	✓	✓*	$2^{-8}$ -secure	✓	✓	✓	✗
VeCure [62]	✓	✓	✗	✗	✓	✓	✗
Woo-Auth [12]	Basic: ✓ Enhanced: ✗	✓	✗	✗	✓	✓	✗
LeiA [7]	✗	✓	✗	✗	✓	✗***	✓
vatiCAN [63]	✗	✗	✓**	✗	✓	✗***	✓
LiBrA-CAN [64]	✓	✓	✓	✓	✓	✓	✓
VulCAN [65]	✗	✓	$2^{-6}$ -secure	✗	✓	✗***	✓
TOUCAN [66]	✓	✓	$2^{-24}$ -secure	✗	✓	✓	Not needed****
AuthentiCAN [67]	✓	✓	✓	✓	✓	✓	✓
S2-CAN [39]	✓	✓	✗	✗	✓	✓	✓

Gen. Int. = General Internal; Sel. Int. = Selective Internal; Ext. = External.

\* Lower security strength compared to the other protocols.

\*\* If proper group design.

\*\*\* Possible delays due to non-atomic authentication.

\*\*\*\* No use of nonces/counters.

transmission of the two messages as the CAN+ message is sent during the standard CAN transmission.

**Car2X** employs data segmentation to transmit cryptographic keys and data packets. Thus, for  $n$  segments, there is at least  $n$  times the packet transmission and handling time with respect to a single packet. Furthermore, other ECUs might interpose in between during packet transmission due to arbitration, delaying packet reception. Although authors state that for some applications requiring a less strict security level, MAC truncation can be adopted because an even level of security must be guaranteed across the whole CAN bus, atomic authentication is not guaranteed. For these reasons, Car2X does not comply with this requirement.

**LinAuth** attaches the employed counter and all the MACs for authentication to the authenticated frame. However, since the protocol expects that a transmitting ECU computes a MAC value for each of the interested ECUs, the number of MACs computed may exceed the available space in a single CAN frame payload. Since there is no mention of how to handle such a scenario in the protocol specification, we assume the easiest and best scenario possible. Hence, we assume that in case the MACs do not fit into the authenticated message, the ECU immediately transmits a second frame in which the payload contains the remaining MACs. To prevent other ECUs accessing the bus in between the transmission of the two messages, one of the free bits of the CAN protocol can be used to signal that a following message will be transmitted carrying the remaining MAC values. In doing so, we let the protocol comply with the atomic authentication requirement.

**MaCAN** ensures atomic authentication by incorporating several values in the authentication request message, such as the

desired message, the prescaler value, and the cryptographic MAC computed using the sender's key. The recipient ECU, upon receiving the request, processes it as a complete authentication request. This ensures that the authentication process is atomic, with no intermediate or partial stages.

**VeCure** does not attach the MAC value to the authenticated message but transmits it in a subsequent frame. However, the protocol defines that the authentication frame is transmitted immediately after the one carrying the data. Hence, if a receiving ECU sees a CAN message with an  $ECU_{ID}$  that is in a trusted group (thus will be authenticated), it will not attempt to access the bus due to the immediate transmission of the authentication message. This guarantees that no ECU can transmit in between the transmission of the two messages. VeCure authentication procedure can thus be considered atomic.

**LCAP** uses a hash chain mechanism to authenticate the exchanged frames. Chain values are attached to the same frame to authenticate, thus making the protocol compliant with the atomic authentication requirement.

**Woo-Auth** expects two possible operation modes for authentication. If the protocol follows the *basic method*, it complies with this requirement as the authentication tag is attached to the frame. Otherwise, if using the *enhanced method*, the tag is split into two messages, thus leading to possible interruptions and delays between their transmission, making it not compliant.

**CaCAN**, **LiBrA-CAN** and **TOUCAN** authenticate the exchanged frames by attaching the corresponding HMAC value to the same message they are authenticating, thus letting the receivers be able to verify the authenticity of the message immediately. Therefore, the protocols comply with this re-

quirement.

**LeiA**, **vatiCAN** and **VulCAN** do not comply with the atomic authentication requirement since they expect to transmit the authentication tag in a separate message with respect to the authenticated data. Moreover, all protocols expect to transmit the authentication message with a different ID with respect to the corresponding message it wants to authenticate. As a drawback, if the number of data messages is greater than  $2^{|ID\ bits|-1}$ , where  $|ID\ bits|$  is the number of bits used for the ID, then not all the messages can be authenticated.

**AuthentiCAN** does not require using any specific authentication tag as this is achieved thanks to the uniqueness of the private key used for encryption. Therefore, if a message is decrypted, the transmitter uses the correct private key, proving its legitimacy. The protocol complies with this requirement since a correct encryption grants authentication.

**S2-CAN** provides authentication through a correct encoding of the exchange payload, thus providing atomic authentication.

### B. Replay Attack Resistance

**CANAuth** and **LeiA** make use of a fresh element for each security object computed (i.e., session key, session key signature, payload signature, MAC). Whenever a message with a nonce that has already been used is received, it is discarded a prior.

**Car2X** does not mention any usage of counter or nonces in the MAC computation. Furthermore, while the protocol uses session keys, they last up to 48 hours, giving plenty of time for an attacker to flood the CAN bus with previously sniffed authenticated messages. Thus, Car2X is not resistant to replay attacks.

**LinAuth** uses an increment counter for MAC computation, thus protecting against replay attacks. In the case of a replayed message, since the frame only carries the least significant bits of the counter, the verifier would increase the most significant part. This will result in a different MAC value than carried by the message, thus leading to message rejection.

**MaCAN** utilizes timestamps for message freshness in unidirectional, event-driven scenarios where replay attacks are a concern. It introduces a Time Server (TS) to ensure synchronized time signals, transmitting timestamps regularly and offering on-demand authenticated timestamps. However, replay attacks are still a threat when using group keys since all group members cannot verify the challenge sent by the ECU to the TS in the authenticated time distribution protocol.

**LCAP** is resistant to replay attacks thanks to the hash values attached to each message. Indeed, replaying any previously authenticated message leads the receiver to reject it, as the authentication check will fail. The only non-authenticated message happens during the *Channel Setup* phase with the transmission of the 32-bit nonce from the receiver. It is worth noting that this message cannot be exploited for a replay attack as the sender will perform the hash computation over a non-valid nonce, forcing the legitimate receiver not to accept the transmitted parameters.

**VeCure** is resistant to replay attacks since, for a given  $ECU_{ID}$ , the MAC computation depends on the triple

$(c_s, c_o, c_m)$ , which is unique during the vehicle lifetime. Thus, the receiver will reject any previously authenticated message. It is worth mentioning that in our analysis, we only consider the authenticated messages. Messages from ECUs in the low-trust group are not authenticated. Nevertheless, we can provide replay attack resistance in this latter case by simply introducing authentication for the low-trust group.

**Woo-Auth** is resistant against replay attacks. If during the *Initial Session Key Distribution* phase, a malicious ECU tries to replay an old  $(S, MAC_1)$  message, the receiving ECU would not accept it. Indeed, the  $MAC_1$  value also depends on the previously exchanged  $R$  value, which differs from the one in use. Therefore, because of the security assumptions for the HMAC algorithms, the expected MAC value will differ from the received one, and the message will be discarded. A similar reasoning applies to the *Session Key Update* phase. Lastly, messages are authenticated using a counter, which assures freshness and replay attack resistance.

**CaCAN** uses counter values for HMAC computation, avoiding that two equal messages return the same hash value. However, it must be noticed that only the 8 least significant bits of the HMAC are considered for authentication, and only the least significant part of the counter is exchanged. Therefore, an attacker may have the chance to succeed in replaying a message. Two equal messages may authenticate with two different counters but with the same least significant part, return different HMAC values but having the same 8 least significant bits. In such a scenario, the central node would classify the message as legitimate. We are aware that this scenario is very unlikely. However, we believe that 8 bits are not enough to guarantee a security level and resistance that can be considered absolute, as in previous protocols. Therefore, we have to highlight such a difference in resistance strength.

**LiBrA-CAN** uses counters attached to each message to ensure freshness in messages and resistance against replay attacks.

**VulCAN** uses an increasing counter or nonce  $c_i$  as a source of freshness in the MAC computation (where  $i$  is the message identifier). Nonces are always different when using the same key and associated data. Thus, the authors provide a detailed description of the nonce initialization and resynchronization process in case a packet loss occurs.

**vatiCAN** uses a counter value to provide freshness for MAC computation. However, the protocol may still be vulnerable to replay attacks. Indeed, because no session key is used, a payload with the same  $ID_i$  and  $c_i$  will always return the same  $MAC_i$ . Therefore, if for message  $ID_i$ , the counter used was  $c_i = t$ , but the Nonce Generator broadcasts a value  $g < t$ , an attacker will successfully replay all messages  $ID_i$  such that  $g < c_i < t$ .

**TOUCAN** makes use of the Chaskey algorithm to compute the MACs. Although the Chaskey algorithm does not use any fresh element for the computation, its design proves it is resistant to replay attacks [69].

**AuthentiCAN** attaches a fresh element to the exchange message's payload, thus avoiding possible replay attacks.

**S2-CAN** uses a counter value both in the Handshake and Operation phases, thus defending against possible replay attacks.

### C. Masquerade Attack Resistance

**CANAuth** may be vulnerable to both internal masquerade attacks (*General* and *Selective*) if the group keys approach is adopted and groups are poorly designed. Instead, if the approach is adopted consistently to what we defined in Section VII, the protocol will be resilient to *General-internal* masquerade attacks. The attacker must first learn either the master key or session key to authenticate correctly a forged message. This is, however, not possible as  $K_i$  is assumed to be stored in a secure memory. At the same time,  $K_{s,i}$  is computed using an HMAC algorithm that assures a very high level of security. However, the protocol remains vulnerable to *Selective-internal* masquerade attacks. Lastly, the protocol is resilient against external masquerade attacks but vulnerable to the *Selective-internal* masquerade attack.

**Car2X** is resistant against internal and external masquerade attacks thanks to its use of a Key Master (KM) entity, which handles the key distribution. We assume the KM cannot be compromised since the authors define it as an external entity. External masquerade attacks can be prevented since external devices do not own a key pair  $k_{s,g}$  and  $k_{s,v}$  needed to communicate with the KM. Internal masquerade attacks, regardless of their type (i.e., *General* or *Selective*), are prevented since the KM authorizes communication between an ECU  $e_1$  and a group  $g_x$  based on predefined policies.

**LinAuth** is secure against both internal and external masquerade attacks. Since the protocol employs a pairwise key approach, a rogue ECU would need to learn all the necessary keys to successfully compute the MACs for all the interested ECUs. On the other hand, if a group keys approach were employed, groups must be carefully designed; otherwise, this may open to internal masquerade attacks vulnerability.

**MaCAN** authors provide a detailed analysis of the risk of guessing the signature during the lifetime of a key. Indeed, considering an attempt each 20 ms and a key lifetime of 48 hours, the total number of attempts becomes 8640000. Therefore, with a 32-bit MAC, internal masquerade attacks have a 0.002 (1 in 500) chance of occurring with a brute force attack. However, an intrusion detection system can easily mitigate attempts at a high frequency on the bus. Instead, using a symmetric key to compute the MAC, *Selective-internal* masquerade attacks can still occur.

**LCAP** may be vulnerable to a masquerade attack carried on by an internal attacker. Consider transmitter A and receivers B, C, and D. It should be noted that after the *Channel Setup* phase, all the receivers will have the same session and HMAC keys and the hash chain exchanged values. Hence, we can say that the *Channel Setup* phase actually sets up all the needed parameters to receive messages from that transmitter. As a result, a compromised ECU-B could correctly encrypt a forged message and send it to ECU-C on behalf of transmitter ECU-A. However, to have the message accepted, the attacker must guess the correct  $M_j^i$ , which is assumed impossible from the previous values thanks to the security properties granted by the HMAC algorithm. In their proposal, authors assume hash values of 16 bits. Therefore,  $2^{16}$  different values are possible. The attacker can discard the already used values, making the

attack success probability higher if performed close to the end of the hash chain. We derive that the security of LCAP against a masquerade attack carried on by an internal attacker at step  $k$  is:  $(2^{16} - n)^{-1}$ . On the other hand, the protocol is secure against external attackers as they first have to learn the session or master key, which is assumed to be computationally difficult or not possible because it is stored in protected memory.

**VeCure** is resistant against external masquerade attacks since an external device would need to learn the secret key of the trust group of the target ECU. However, this is not possible thanks to the security assumption for the hash and BME functions. On the contrary, VeCure is vulnerable to internal masquerade attacks: a compromised ECU in trust-group  $i$  could successfully authenticate messages on behalf of any ECU belonging to group  $j \leq i$ . Indeed, the rogue ECU would have the proper keys and the session counter (which is the same for all the ECUs). Moreover, it could monitor the bus to retrieve the message counter  $c_m$  of the target ECU and could align with the overflow counter  $c_o$  by (i) trials and errors from previous authenticated messages or (ii) sniff on the bus since the beginning of the session. Therefore, the rogue ECU would have all the elements to authenticate messages on behalf of another node.

**WooAuth** is completely vulnerable to an internal masquerade attack. It must be noticed that to verify the received messages, the  $K_{EK}$  between sender and receiver must be the same. The key is derived from the KDF function, which in turn depends on  $ID_{GW}$  (that is the same for all ECUs), and the random seed  $S$ , provided by GW. Therefore, seed  $S$  must be the same for all ECUs. We conclude that all ECUs in the network will have the same session keys. As a result, any compromised ECU can perform the  $AES - 128$  encryption correctly. The compromised ECU can thus make trials to understand, given the legitimate exchanged message  $M$ , which is the counter value returning the correct  $C$  value, and consequently the MAC. Once this operation is performed successfully, the attacker is aligned with the counter value and can correctly authenticate future messages on behalf of the target ECU. On the other hand, external attackers must first learn the session keys to perform the operations described above. Still, since the security property assures the secrecy of the  $KDF$  process, we conclude the protocol resists external masquerade attacks.

**CaCAN** behaves similarly to WooAuth, and an analogous analysis can be made. Indeed, even though the counter is not fully transmitted, the attacker must sniff an exchanged packet to be aligned with its least significant part. Consequently, an internal attacker may forge a frame with the correct  $LC_i$  value. At this point, they must guess the 8 bits of the HMAC value correctly. It is worth noting that no advantage can be taken from previously exchanged authenticated frames, as the authentication value is part of the full HMAC value. Thus, repetitions are not possible since the full hash value is guaranteed to be almost unique. To conclude, the protocol offers a  $2^{-8}$ -security level against *General-internal* masquerade attacks. The protocol is instead secure against *Selective-internal* attacks, as the only entity that verifies the authentication tags is the Monitor Node. It is also secure against external attacks since the key should be learned first. However, this is assumed

to be impossible.

**LeiA** is completely vulnerable to an internal masquerade attack. In fact, it should be noticed that since the *Setup* phase only depends on a common security parameter  $\eta$ , every ECU will possess the long-term keys  $K_i \forall ID_i$ . Therefore, an ECU can compute  $K_i^s$  for any  $ID_i$  and authenticate any message successfully. This clearly leads to a masquerade attack vulnerability. On the other side, the protocol is secure against external attacks, as the attacker must first learn either the long-term key or session key, which is not possible because of assumptions for the (i) secure storage and (ii) security provided by the session key derivation function.

**vatiCAN** masquerade resistance analysis is analogous to the one we did for CANAuth. If the group keys are well-designed, the protocol resists masquerade attacks. Otherwise, a vulnerability flow is left against the internal masquerade attacks. Resistance against external attacks is guaranteed.

**LiBrA-CAN** is resistant against both internal and external masquerade attacks. Each ECU is provided with the keys of the groups it belongs to. Thus, if  $ECU_j$  wants to impersonate  $ECU_i$ , it will need all the keys in  $\bar{K}^i$ , but this is not possible by group construction because this would mean that  $ECU_j$  has to belong to the same groups  $ECU_i$  is part of.

**VulCAN** does not provide specific mechanisms for addressing masquerading attacks. Anyway, by using a symmetric key for the computation of the 64-bit MAC, it provides  $2^{-6}$ -security against *General-internal* masquerade attacks. *Selective-internal* masquerade attacks can still occur since an attacker can send a malicious CAN frame and subsequently compute its MAC, assuming the attacker has also gained knowledge on the counter. Since the nonce increases in each sent message, the leakage of one counter, or the knowledge of the starting value, allows the attacker to infer it. Furthermore, the authors explicitly state that nonce values are not confidential and that receivers can accept MACs generated with any nonce that is strictly higher than the previously authenticated nonce value.

**TOUCAN** protocol specifications do not directly define how the keys are managed and shared, whether all the ECUs have the same key, or if keys are ECU-specific or message ID-specific. Therefore, an analysis for each of these three scenario must be considered. If just one single key is used, the protocol would be vulnerable to an internal attacker. This also holds in the second case, in which we had ECU-specific keys: it is reasonable to assume, indeed, that all the ECUs are equipped with the keys of (almost) all the others since they would need that for authentication verification. This situation can be reduced to the case in which just one key is used, thus making the protocol vulnerable. The last possible case is the only one in which the protocol is not vulnerable: to successfully authenticate a message, a malicious ECU should first learn the corresponding key or guess the correct authentication tag. The probability of this happening is  $2^{24}$ , which is very low. Thus, we can consider the protocol resistant against *General-internal* masquerade attacks. For the rest of our analysis, we will assume TOUCAN will adopt this last solution as it is the most secure and reasonable. This is also reflected by the security level reported in Table V. However, the protocol is

instead fully vulnerable against a *Selective-internal* attacker in any of the three presented scenarios. TOUCAN is resistant to external attacks.

**AuthentiCAN** is masquerade attacks resistant since an attacker must learn the target ECU's private key for a correct authentication. However, this is assumed not possible for asymmetric encryption's security properties.

**S2-CAN** is completely vulnerable against an internal masquerade attack for the same reasons presented during the analysis of LeiA protocol. Indeed, after the handshake phase, all the ECUs will be provided with (i) the same global encoding parameter  $f$ , and (ii) all the parameters of the other ECUs in the network. Therefore, a compromised ECU can correctly encode (thus authenticate) a message on behalf of another node. On the other hand, the protocol is resistant to external masquerade attacks.

#### D. Hard Real-Time Compliance

**CANAuth** is compliant with the *atomic authentication* requirement, making the only overhead introduced given by the generation and verification of the authentication message. As  $sigM_i$  is computed with an HMAC algorithm that is assumed to be fast, the protocol can be considered hard real-time compliant.

**LinAuth** only requires a counter comparison and MAC computation for authentication, which are considered fast operations. However, it is worth noticing that an ECU has to compute several MACs, one for each of the interested receiving ECUs. This may lead to the case for which the computation of all the MACs introduces a significant time delay, not ensuring hard real-time properties. Using a group keys approach (which the protocol allows for) can reduce the number of keys and MACs to compute, thus making the protocol hard real-time compliant. On the other hand, as already discussed, when using a group keys solution, particular attention must be paid to group design as this could open to masquerade attacks.

**MaCAN** truncates the MAC signatures down to 32 bits to fit into the frame payload, allowing for *atomic authentication* and protecting real-time capabilities. On the other hand, it is worth noting that Nowdehi et al. [30] states that MaCAN does not have an acceptable overhead and thus would not comply with the hard real-time requirement. In doing so, Nowdehi et al. [30] cite a work of Vasile et al., which carries out a performance analysis of broadcast authentication protocols in for CAN [100]. However, this work (Vasile et al.) focuses on implementing protocols in FlexRay and CAN-FD, while the authors of MaCAN explicitly state that their focus is on the standard CAN standard. To conclude, MaCAN satisfies the hard real-time requirements according to our definition.

**LCAP** complies with the *atomic authentication* requirement attaching the hash value to the authenticated frame. As the time needed for (i) hash chain computation, (ii) setup, and (iii) synchronization can be considered amortized, the only significant time overhead is due to the hash verification. This is considered fast because of the HMAC algorithm. Therefore, the protocol satisfies the hard real-time requirement.

**Woo-Auth** uses *AES*–128 encryption and hashing during the frame authentication process. Both algorithms are considered fast [32]. Hence, the protocol offers hard real-time capabilities since the session key distribution and update phases can be considered amortized in time.

**CaCAN** uses an HMAC algorithm for authentication, which is considered fast by assumptions. The protocol is thus hard real-time compliant.

**VeCure** satisfies the *atomic authentication* requirement. Besides, thanks to the MAC design, the computation of the *hash* value can be carried out offline, removing the most demanding operation from the online MAC computation and verification process. In conclusion, VeCure offers hard real-time capabilities.

**LeiA** is not hard real-time compliant as the authentication tag is transmitted in a separate message than the data frame. This may raise issues for the processing of the messages. Suppose an ECU with a high priority accesses the bus after sending the data message. In that case, the authentication message would be delayed, making the receiver wait for the MAC, unable to verify and process the data received timely.

**vatiCAN** hard real-time compliance analysis follows the same reasoning of LeiA: due to the non-atomic authentication, there can be possible delays in receiving the authentication tag. Therefore, we classify the protocol as not hard real-time compliant.

**LiBrA-CAN** may raise some doubt about its hard real-time execution because of the computation of multiple MACs for a single message. However, M-MAC computation is still fast enough to ensure hard real-time capabilities, as shown by simulation results [64].

**VulCAN** considers real-time compliance as a system-level guarantee that is provided by the protocol. Indeed, while the authors do not consider the issue of denial-of-service attacks, they still acknowledge the safety-critical functionality provided by the data exchanged on the bus. To provide this requirement, the MAC payload is truncated to adhere to AUTOSAR specifications. However, due to the non-atomic authentication, several delays might occur, which can alter the normal functionality of an ECU.

**TOUCAN** attaches the MAC to the authenticated message and encrypts it with AES-128. The only overhead introduced is then due to the two cryptographic mechanisms. As both AES-128 and Chaskey are considered to be fast algorithms, we conclude that the protocol is hard real-time compliant.

**AuthentiCAN** may raise concerns about the time overhead it may introduce due to its employment of asymmetric encryption. However, simulations have shown that the encryption overhead is minimal [67]. Hence, because of this, the fact that ECUs are becoming increasingly powerful (reducing computational times), and that public key broadcasting and nonce list transmission phases can be considered amortized in time, the protocol can be considered hard real-time compliant.

**S2-CAN** is hard real-time compliant as the payload encoding is a fast operation (as also shown by experiment results [39]), and the session update phase can be considered amortized in time.

### E. Resynchronization Capability

**CANauth** and **S2-CAN** provide no explicit mechanism for resynchronization. However, since nodes will accept the authentication tag if the received nonce is greater than the stored one, even in case of losses, future legitimate messages can still be successfully authenticated.

**MaCAN** uses a Time Server (TS) to prevent issues that may occur when ECUs maintains the time signal using an internal counter. The message the TS sends exclusively comprises a 32-bit timestamp transmitted regularly. Each node can receive and compare the time signal to its internal clock. If the signals differ significantly or resynchronization is needed, such as after an ECU reboot, an authenticated time signal can be requested.

**LCAP**, **LeiA** and **AuthentiCAN** provide explicit procedures for resynchronization, being then robust to losses and letting ECUs be aligned to the last exchanged parameters and thus guaranteeing the correct functioning of the system.

**Woo-Auth** and **VeCure** do not provide any resynchronization mechanism. Therefore, if a message is lost, the ECU will not be aligned with the counter(s) value(s) and will not accept future messages because of verification failure until the next session update and counter(s) reset.

**LinAuth** does not provide any counter resynchronization mechanism in case of message losses. However, we may notice that such a mechanism is not needed, as the condition for message acceptance is not counter matching (i.e., received counter = stored counter + 1) but counter increasing (i.e., received counter > stored counter). This allows ECUs to correctly verify the authentication even in case of message losses.

**CaCAN** does not offer any resynchronization mechanism as well. Thus, if the monitor node misses one frame, all the following messages will be blocked by raising an error due to counter-value misalignment. This situation will last until the *UC* value of the counter is increased since the *OLC* value will be set as the received *LC*.

**vatiCAN** provides a means to periodically align all the ECUs to the same counter values thanks to the Nonce Generator node. Thus, if a node remains out of synchronization, it cannot accept authenticated messages for a limited time.

**LiBrA-CAN** assumes a resynchronization mechanism is in place, but possible implementations are not mentioned.

**VulCAN** provides nonce resynchronization by “vulcanizing” the process employed by vatiCAN. In this implementation, 16-bit nonces are encoded in the extended CAN identifier, allowing for the re-establishment of session keys or counters.

**TOUCAN** does not need any nonce or counter for authentication, which means that no resynchronization mechanism is needed: if a message loss occurs, it will not affect the possibility of authenticating the following messages.

## IX. EVALUATION OF OPERATIONAL CRITERIA

In this section, we present an analysis based on the operational features previously defined, which are backward compatibility (Section IX-A), group keys (Section IX-B), session keys (Section IX-C), and storage overhead (Section IX-D).



Again, these comparison criteria do not change the security capabilities of the protocols. Rather, they may increase their security level and give a more complete overview of their features. Table VI summarizes what is discussed in this section.

#### A. Backward Compatibility

**CANAuth** relies on CAN+, requiring transceivers to support up to 60 Mbit/s data rate. Therefore, the protocol is backward compatible for ECUs that supports such data rate. Otherwise, ECUs not equipped with such transceivers will require a hardware modification or substituting.

**Car2X** is not backward compatible since it requires the implementation of the Key Master module to function properly. The authors define this component as an external device and, as such, cannot be implemented on an existing device in the bus.

**LinAuth**, **LCAP**, **VeCure**, **Woo-Auth**, **LeiA**, **vatiCAN**, **LiBrA-CAN**, **VulCAN**, **TOUCAN**, and **S2-CAN** are fully backward compatible as they only require software update to the ECUs to implement the authentication protocol.

**MaCAN** is also backward compatible with the standard CAN frame format but also needs the implementation of the Time Server. As stated by the authors, this component can be added to an existing ECU. Therefore, since it requires introducing new specific hardware, the protocol is not backward compatible according to our definition.

**CaCAN** is not backward compatible as it requires adding a task-specific Monitor Node adopted with specific hardware (special CAN controller) [61]. Conversely, no hardware modification is needed for all the other standard ECUs.

**AuthentiCAN** is backward compatible as it relies on CAN-FD protocol, and no further hardware modification is needed.

#### B. Group Keys

**CANAuth** and **vatiCAN** use a group keys approach. However, as already discussed, if this solution is not properly designed, the protocols will become fully vulnerable to an internal masquerade attack.

**Car2X** uses a group key approach thanks to the predefined policies enforced by the KM. Indeed, the KM must first authorize ECUs that request to send messages to a group.

**LinAuth** allows a group keys approach in which ECUs are assigned to groups, and all share the same key. However, as already discussed, the groups must be carefully designed to avoid open to internal masquerade attacks.

**MaCAN** makes use of group keys. However, using those keys allows the attacker to perform replay attacks during the authenticated time distribution protocol.

**LCAP** does not use any group keys approach. Rather, it lets every receiver possess all the security parameters for each sender, which may lead to possible masquerade attacks, as previously discussed.

**Woo-Auth** makes use of no group key approach and, as well as **LCAP** protocol, it makes all the ECUs share the same session keys, opening them to the internal masquerade attack vulnerability previously discussed.

**VeCure** protocol is based on trust-level group ECUs partitioning the ECUs, and for each group a key is defined. This represents a group keys approach. However, as discussed in the *masquerade attack resistance* section, **VeCure**'s group design opens to masquerade attacks.

**CaCAN**, **VulCAN**, **TOUCAN**, and **S2-CAN** do not use any group keys approach.

**LiBrA-CAN** makes use of group keys securely.

**LeiA** uses one key for each message ID, thus being consistent with our definition of group keys.

**AuthentiCAN** does not use group keys because of the asymmetric encryption.

#### C. Session Keys

**CANAuth**, **Car2X**, **MaCAN**, **LeiA**, and **LiBrA-CAN** make use of session keys, thus improving their security in the case in which a key is compromised. Possible damages are then limited for the key lifetime, after which the new key must be learned.

**LCAP** and **Woo-Auth** make use of session keys as well. However, we want to specify that because of their specifications, this solution only has beneficial effects against external attackers. At the same time, no advantage is gained against internal attackers as all the receiver ECUs possess all the session keys.

**LinAuthCaCAN**, **VeCure** and **vatiCAN** do not make use of any session key approach.

**VulCAN** uses short-term session keys, which are refreshed at system boot time or whenever the session counter  $c_i$  overflows. These session keys are based on the long-term pre-shared secret and a larger epoch counter, safely allowing nonces to start from zero at each session.

**TOUCAN** does not make use of session keys. Moreover, since Chaskey does not make use of nonces for the MAC computation, authors state that to avoid attacks with a practical complexity of off-line permutation evaluations, the total number of blocks to be authenticated with the same key shall be at most  $2^{64}$  [69]. Even though this is a very large number, we must consider that as the number of electronic components continuously increases, this number may not be sufficient anymore to cover a vehicle's lifetime.

**AuthentiCAN** does not make use of session keys. However, considering the security behind asymmetric key generation, we recognize that using session keys in an end-to-end communication protocol does not offer any significant improvement in terms of security. Rather, this could increase some non-necessary overhead and power consumption.

**S2-CAN** does not properly use session keys in the strict meaning of the term. However, payload encoding is performed according to the global encoding parameter  $f$ , which can be considered a key. As a result, since this is updated at each new session, we can consider that the protocol uses session keys.

#### D. Storage Overhead

Since protocols do not specify the number of bits reserved for all the parameters used for authentication, and given that part of the additional storage required to ECUs depends on

TABLE VI: Operational criteria comparison of the different protocols.

Protocol	Backward Compatibility	Group Keys	Session Keys	Storage Overhead
CANAuth [13]	✓*	✓**	✓	$332 \cdot n\_groups + 29 \cdot x \cdot \log_2(n\_groups)$
Car2X [58]	✗	✓	✓	$(86 + 256 + 128) \cdot 2n$
LCAP [14]	✓	✓	✓	$164 \cdot (n - 1) + 96 \cdot n + \lambda \cdot (36 + 16 \cdot x\_send) + 16 \cdot \sum_{i=1}^{n-1} x\_send_i$
LinAuth [59]	✓	✓**	✗	$L_K \cdot (n - 1) + L_{ID} \cdot \left( \sum_j^{x\_send} x_{receive_j} + 1 \right) + L_c \cdot (x + x\_send)$
MaCAN [60]	✗	✓**	✓	$L_K \cdot n$
CaCAN [61]	✗	✗	✗	$L_{ss} + 512$
VeCure [62]	✓	✓	✗	$128 \cdot (i - 1) + (16 + L_{oc}) \cdot \sum_{j=2}^i g_j + 24$ where $i = 2, \dots, n\_groups$
Woo-Auth [12]	✓	✗	✓****	$2 \cdot L_{MK} + L_{SK_s} + (L_c + L_{ID}) \cdot n$
LeiA [7]	✓	✓	✓	$328 \cdot  ID  + L_\eta$
vatiCAN [63]	✓	✓****	✗	$(128 + L_c) \cdot n\_groups + 29 \cdot (\log_2(n\_groups) \cdot x_{groups} + p)$
LiBrA-CAN [64]	✓	✓	✓	$\binom{n-1}{q-1} \cdot L_K + L_c$
VulCAN [65]	✓	✗	✓	$(L_K + L_{EC} + 32) x$
TOUCAN [66]	✓	✗	✗	$128 \cdot (x + x\_send)$
AuthentiCAN [67]	✓	Not needed	Not needed	$(512 + L_K) \cdot n + L_K$
S2-CAN [39]	✓	✗	✓	$L_K + Q + 48$

\* Only for transceivers supporting up to 60 Mbit/s

\*\* Adoption may open to replay attacks.

\*\*\* Adoption may open to internal masquerade vulnerabilities without proper group design.

\*\*\*\* Effective only against external attackers.

the specific implementation scenario, our analysis will provide the more general analysis as possible. Table VII provides the meaning of the common symbols we use in our analysis between the protocols. Values that are specific to a single protocol will be introduced in the specific analysis.

TABLE VII: Notation for the storage overhead analysis.

Value	Meaning
$n$	Number of ECUs in the network
$n\_groups$	Number of groups when using the group keys approach
$g_i$	Size of the group $i$ when using the group keys approach
$x$	Number of different messages (IDs) the ECU is interested in receiving
$x_{send_i}$	Number of IDs that $ECU_i$ can transmit
$x_{receive}$	Number of different receivers that the ECU considers
$L_{MK}$	Number of bits for the master key
$L_{SK_s}$	Sum of the number of bits of the session keys used
$L_K$	Number of bits used for the secret keys (no distinction between master or session keys)
$L_c$	Number of bits used for the counter
$L_{ID}$	Number of bits used for the ECU-specific identifier
$L_{ss}$	Number of bits used for the stored shared secret
$L_\eta$	Number of bits used for the secret parameter $\eta$
$\lambda$	Number of elements of the hash chain

**CANAuth** requires an additional storage of 128 bits for  $K_i$  and  $K_{S,i}$  each, 24 bits for  $ctrA_i$  and 32 bits for  $ctrM_i$ . This holds for each defined group. Moreover, ECUs must maintain a table that relates the IDs to the corresponding group. The minimum size of the table depends on how many IDs the specific ECU is interested in. Therefore, we have that  $29 \cdot$

$\log_2(n\_groups) \cdot x$  additional bits are required for storing the table (analysis is done in the worst case where the extended identifier is used). To conclude, CANAuth requires a total of  $(128 + 128 + 24 + 32) \cdot n\_groups + 29 \cdot x \cdot \log_2(n\_groups) = 332 \cdot n\_groups + 29 \cdot x \cdot \log_2(n\_groups)$  additional storage bits per ECU.

**Car2X** stores in the HSM of each ECU two keys that carry a header field of 86 bits, key data of 256 bits, and an authentication code of 128 bits, as defined also in [101]. These two keys are  $K_{i,a}$  and  $K_{i,t}$ . Each KM then stores the same two keys for each ECU  $i$  in the group the KM manages. However, since the KM is an additional node in the bus, we do not count its storage in the overhead since we assume that external hardware is equipped with the necessary capabilities. Thus, the additional storage is  $(86 + 256 + 128) \cdot 2n$ .

**LinAuth** requires that each ECU stores a symmetric key with each of the other ECUs in the network, requiring  $L_K \cdot (n - 1)$  bits. Besides, ECUs are required to store their own specific  $ECU_{ID}$  and an ID-table listing for each of the possible message ID they can transmit, the  $ECU_{ID}$  of those nodes interested in receiving the corresponding message. For each ID  $j$  that an ECU can transmit, storing the table requires  $L_{ID} \cdot \sum_j^{x\_send} x_{receive_j}$  bits. Furthermore, for every transmitted and received message ID, the ECU must also store a counter for freshness in MAC computation/verification. This requires  $L_c \cdot (x + x\_send)$  bits. In conclusion, the protocol requires a total of  $L_K \cdot (n - 1) + L_{ID} \cdot \left( \sum_j^{x\_send} x_{receive_j} + 1 \right) + L_c \cdot (x + x\_send)$  additional storage bits for each ECU.

**MaCAN** requires all participating nodes to have secure memory to store the key material. However, since session keys can be requested at any time to the key server, there is no need for

the nodes to permanently store any information other than their long-term key (LTK). However, the paper does not specify the length of the long-term key. Thus, the storage overhead becomes  $L_K \cdot n$ . This is valid whether the ECUs that need to communicate belong to the same group, since in the former case, a common group key can be requested and stored by the group members.

**LCAP** protocol expects that for each couple of ECUs, a 128-bit master key is stored. This will then require  $128 \cdot (n - 1)$  bits. Further, ECUs will have to store their own and one for any other node 80-bit session key, requiring  $80 \cdot n$  bits. The same reasoning is for the HMAC key:  $16 \cdot n$  bits. Moreover,  $2 \cdot 16 \cdot \lambda$  bits are required to store the handshake and channel hash chains, while just a single hash value is required for received messages verification, which leads to  $2 \cdot 16 \cdot (n - 1)$  bits. As last,  $16 \cdot \lambda \cdot x_{send}$  bits are needed for the transmitted messages hash chain, and  $16 \cdot \sum_{i=1}^{n-1} x_{send_i}$  are needed for the received messages. Therefore, the protocol requires a total of  $128 \cdot (n - 1) + 80 \cdot n + 16 \cdot n + 2 \cdot 16 \cdot \lambda + 2 \cdot 16 \cdot (n - 1) + 16 \cdot \lambda \cdot x_{send} + 16 \cdot \sum_{i=1}^{n-1} x_{send_i} = 164 \cdot (n - 1) + 96 \cdot n + \lambda \cdot (36 + 16 \cdot x_{send}) + 16 \cdot \sum_{i=1}^{n-1} x_{send_i}$  bits storage overhead.

**VeCure** protocol introduces the ECU identifier of 8-bits length and a session counter  $c_s$  of 2 bytes. Then, assume having  $n_{groups}$  trust level groups that are enumerated starting from 1, corresponding to the lowest trust level group. An ECU in group  $i$  has to store a 128-bit cryptographic key for each group  $j = 2, \dots, i$  (remember the lowest group does not provide authentication; hence it has no secret key), thus  $128 \cdot (i - 1)$  are needed. Besides, each ECU has to store a 2-byte message counter  $c_m$  and an overflow counter  $c_o$  for itself and each of the ECU belonging to security groups  $j = 2, \dots, i$ . This will then require  $16 \cdot \sum_{j=2}^i g_j + L_{oc} \cdot \sum_{j=2}^i g_j$  bits, where  $L_{oc}$  is the length of the overflow counter since it is not specified in VeCure protocol. To conclude, VeCure requires each ECU belonging to group  $i = 2, \dots, n_{groups}$  to handle  $128 \cdot (i - 1) + 16 \cdot \sum_{j=2}^i g_j + L_{oc} \cdot \sum_{j=2}^i g_j + 16 + 8 = 128 \cdot (i - 1) + (16 + L_{oc}) \cdot \sum_{j=2}^i g_j + 24$  additional bits for authentication.

**WooAuth** assumes that the gateway ECU has higher computing power than standard ECUs, so our analysis will focus on the standard ECUs. Moreover, protocol specifications do not give any information about the length of the master and session keys or the counter values. Therefore, we will perform a very general analysis. Each  $ECU_i$  has to store its own  $ECU_{ID}$ , its long-term key  $K_i$ , the gateway key  $K_{GW}$ , and all the other session keys  $(K_E, K_A, K_{EK}, K_G, K_U)$ . Besides, ECUs also has to store its counter and one for each other ECU. Since the counter is ECU specific, there must be maintained a table associating it to the corresponding ECU:  $(L_c + L_{ID}) \cdot n$ . This results in a total of  $2 \cdot L_{MK} + L_{SKs} + (L_c + L_{ID}) \cdot n$  additional storage bits.

**CaCAN** assumes the participation of a special Control Node, for which it is reasonable to assume it is provided with all the needed memory storage and computing power. Hence, our analysis will be based solely on the standard ECUs. A standard ECU must store the shared secret of length  $L_{ss}$ , the 512-bit

encryption key, and a 32-bit counter value. This requires a  $L_{ss} + 512 + 32$  additional storage bits.

**LeiA** protocol expects that each ECU stores two 128-bit keys (master and session), one 56-bit epoch value, and one 16-bit counter for each message ID. Besides, each ECU has to store its security parameter  $\eta$ . Assuming that  $|ID|$  different message IDs are exchanged in the network, each ECU will have to store a total of  $2 \cdot 128 \cdot |ID| + 56 \cdot |ID| + 16 \cdot |ID| + L_\eta = 328 \cdot |ID| + L_\eta$  additional bits.

**VatiCAN** expects that each ECU stores a 128-bit key for each message it is interested in or for each group it is part of, and a  $L_c$  bits for the counter. Moreover, since the possibility of the group key approach, a table that relates the IDs with the corresponding group must be maintained within each network node. Hence, following the same analysis we did for CANAuth,  $29 \times \log_2(n_{groups})$  additional bits are required for this purpose. Lastly, since not every ID is authenticated, a table of critical messages must be maintained. To conclude, given  $x_{groups}$  the number of groups the ECU is interested in, and  $p$  the number of protected IDs, the protocol requires an additional amount of  $128 \cdot n_{groups} + L_c \cdot n_{groups} + 29 \times \log_2(n_{groups}) \cdot x_{groups} + 29 \cdot p = (128 + L_c) \cdot n_{groups} + 29 \cdot (\log_2(n_{groups}) \cdot x_{groups} + p)$  storage bits.

**LiBrA-CAN** expects that each ECU is provided with an encryption key for each group it belongs to and a personal counter value. Hence, knowing that each ECU will belong to  $\binom{n-1}{g-1}$  groups, each ECU is required of  $\binom{n-1}{g-1} \cdot L_K + L_c$  additional bits storage capacity.

**VulCAN** uses 128-bit session keys for each connection. The session keys are generated by both parties based on the long-term pre-shared secret and a larger epoch counter. The session keys are not stored on every ECU but rather generated by the Attestation Server (AS) and distributed to all Protected Modules (PMs) during load-time attestation. However, PMs are hosted in each ECU as a protected software module. It also uses a 32-bit nonce for each authenticated connection. By defining  $L_K$  as the length of the long-term pre-shared secret and  $L_{EC}$  as the length of the epoch counter, the additional storage required is  $(L_K + L_{EC} + 32) \cdot x$ .

**TOUCAN** storage overhead analysis will consider the message ID-based keys scenario since it is the only situation for which the protocol offers *General-internal* masquerade attack resistance properties. Hence, given  $x$  as the number of IDs an ECU is interested in, and  $x_{send}$  the number of IDs the ECU can transmit, recalling that the protocol uses 128-bit keys, the total amount of additional storage an ECU must support is  $128 \cdot (x + x_{send})$ .

**AuthentiCAN** protocol specifications suggest defining a nonce length of 8 bits with a total list length such that it fulfills the payload size. CAN-FD protocol offers up to 64 bytes of payload size. Thus, we derive that a nonce list can contain up to 64 nonces. A list is kept for communication with each node in the network. Additionally, an ECU must store its private and public keys and all the other nodes' public keys. Hence, an ECU is required to maintain a total additional memory storage of  $8 \cdot 64 \cdot n + L_K + L_K \cdot n = (512 + L_K) \cdot n + L_K$  bits.

**S2-CAN** protocol specifications require each ECU to store a

shared symmetric key (whose length is not given), the 3-byte global encoding parameter  $f$ , the 1-byte  $int\_ID_j$  parameter, the  $Q$ -byte  $pos_{int,j}$  parameter, and a 2-byte counter  $cnt_j$ . This results in a total amount of  $L_K + 24 + 8 + Q + 16 = L_K + Q + 48$  additional storage bit requirement.

## X. SECURITY CLASSIFICATION

From the analysis above and relative summary tables, we notice that among the security requirements, the most discriminating one is the masquerade attack resistance. Table VIII presents our ranking of the protocols resulting from our security analysis. The ranking divides protocols into four categories, which are *secure protocols* (Section X-A), *partially secure protocols* (Section X-B), *non-secure protocols* (Section X-C), and *non-suitable protocols* (Section X-D).

TABLE VIII: Protocols' ranking.

Security Level	Protocols
Secure Protocols	1) LiBrA-CAN [64] 2) AuthentiCAN [67] 3) LCAP [14] 4) CaCAN [61] 5) LinAuth [59]
Partially Secure Protocols	6) TOUCAN [66] 7) CANAuth [13] 8) MaCAN [60]
Non-Secure Protocols	9) S2-CAN [39] 10) Woo-Auth [12] 11) VeCure [62]
Non-Suitable Protocols	12) VulCAN [65] 13) LeiA [7] 14) Car2X [58] 15) vatiCAN [63]

### A. Secure Protocols

In this class belong protocols that satisfy all the security requirements. However, one may notice that CaCAN does not satisfy the *resynchronization capability* requirement. Since we believe designing a proper one is not very demanding, we decide to include CaCAN in this class anyway. In particular, what discriminates these protocols from the *almost secure protocols* class is the resistance against *Selective-internal* internal masquerade attacks. Therefore, we conclude that approaches based on H-MAC, hash chain, or asymmetric encryption are the most secure in this particular framework of in-vehicle communications. Besides, LinAuth also shows how using pairwise shared secret keys allows one to successfully defend against internal masquerade attacks; however, this may raise issues with possible hard real-time compliance.

We classify LCAP and CACAN protocols one and two steps lower than LiBrA-CAN and AuthentiCAN because they offer weaker resistance against internal masquerade attacks. However, we recall that this lower resistance is given by the fact that the length of the authentication tags has been constrained due to the limited size of standard CAN payloads. Therefore, if protocols were adapted to work on top of CAN-FD, they would benefit from the increased payload size, having

the chance to increase the length of the authentication tag, which results in an increased level of security. Finally, LinAuth is last classified within this group due to the possible issues for hard real-time deriving from the multiple MACs computation due to the pairwise secret keys approach.

### B. Partially Secure Protocols

In this class belong protocols that satisfy all the security requirements apart from the resistance against the *Selective-internal* masquerade attacks. This class of protocols offers a high-security level with the only exception of that particular situation.

We classify TOUCAN with a higher ranking than CANAuth because (i) it is fully backward compatible and (ii) tends to introduce a lower storage overhead. MaCAN instead scores even lower due to the problematic use of group keys that might allow for replay attacks in specific scenarios.

### C. Non-Secure Protocols

In this class, some protocols do not offer protection against internal masquerade attacks. This clearly is a significant security lack that makes the protocol insecure for our scenario. This is because the security elements of the protocols all depend on one or more common security parameters. Thus, these approaches are not a good solution and are not applicable for secure authentication over CAN.

We classify S2-CAN better than WooAuth and VeCure because these two protocols lack of a resynchronization mechanism, and WooAuth is better than VeCure as it tends to require less storage overhead.

### D. Non-Suitable Protocols

These protocols are unsuitable for hard real-time in-vehicle communication applications as they are not secure and do not satisfy the hard real-time and atomic authentication requirements. However, we have to specify that if CAN-FD was considered, all protocols could become hard real-time compliant with respect to our definition since they would take advantage of the much larger payload size. This would, in fact, allow the transmission of the authentication tag together with the authenticated message without the need to split them into two frames. As a result, if this were the case, LeiA and VulCAN would jump into the *Non-Secure Protocols* class, while vatiCAN and Car2X would still be in this class due to their non-compliance with the replay attack resistance requirement.

## XI. LESSONS LEARNED AND FUTURE DIRECTIONS

In this section, we point out the key takeaways and possible future research directions that result from our security analysis concerning the adoption of cryptographic authentication solutions for the CAN-bus in the automotive application.

*Lesson 1.* By looking at the results of our security requirements comparison in Table V and the protocols classification, we can identify the resistance to masquerade attacks as being the most discriminating requirement. In particular,

thanks to our analysis, we highlight that the major focus in the literature for CAN bus cryptographic authentication protocols is protecting against external attackers, as defined in Section V. While all the considered protocols are resistant to external masquerade attacks, only half of them are secure against *General-internal* internal masquerade attacks. Further, just one-third of the considered protocols are fully secure, thus also offering protection against the specific scenario identified by the *Selective-internal* masquerade attack (VII-B).

**Lesson 1** – *Masquerade attack resistance is one of the most important security requirements to ensure protocol security.*

*Lesson 2.* One of the major criticalities in designing an authentication protocol for CAN is making the authentication tag fit into the frame payload. This is due to the limited space in a CAN frame. The most common solutions to face this issue are either a tag truncation or splitting the authentication phase into two messages and embedding the tag in a frame after the data frame. However, as we have extensively discussed throughout our analysis, these approaches lead to a lower security level in the case of tag truncation or to non-atomic authentication and hard real-time issues, in the case of message splitting. On the other hand, these problems can be solved by adopting the CAN-FD protocol as the new standard for in-vehicle communications. Indeed, CAN-FD not only allows for an extended data payload while maintaining an acceptable transmission time and bus load but also offers full backward compatibility.

**Lesson 2** – *Limited space in a CAN frame compromises the security level, and thus CAN-FD should be preferred.*

*Lesson 3.* The maturity of cryptographic authentication schemes for CAN seems promising, as demonstrated by the fact that as a result of our analysis, 5 out of 15 protocols (LiBrA-CAN, AuthentiCAN, LCAP, CaCAN, LinAuth) are perfectly suitable for a real-life implementation offering a very high level of security and reliability. Besides, there are other three protocols (TOUCAN, CANAuth, MaCAN) that offer a high level of security but leave open vulnerabilities only against the very specific scenario of a *Selective-internal* internal masquerade attack.

**Lesson 3** – *To meet security standards, novel authentication schemes should align with the maturity level of the highest-ranking protocols in our assessment.*

*Lesson 4.* The best approaches for secure authentication in CAN are based on (i) hash chains, (ii) asymmetric encryption, and (iii) M-MACs, as they offer full security against the threat model considered in this work. By adopting M-MACs in a broadcast scenario, we gain the advantage that an internal attacker must first learn the (symmetric) secret keys for all the receiving nodes in order to successfully carry out a masquerade attack and correctly generate the ensemble of MAC values. By adopting a hash-chain solution, we rather

gain advantage of the pre-image resistance property guaranteed by the hash function. Therefore, given the hash value  $H_i$ , an attacker cannot guess the following value  $H_{i+1}$  (which computation comes before  $H_i$  during chain development) for correct authentication.

**Lesson 4** – *Most secure approaches rely on hash chains, asymmetric encryption, or M-MACs.*

*Future Research Directions.* Many of the considered protocols designed the authentication procedure without considering the key distribution phase and maintenance, rather than assuming them as a service. However, in real-life scenarios, these are still open issues that harden the adoption of these protocols, although being mature enough, from the authentication security point of view, to be deployed. Future research should focus on implementing new methods and optimizing existing ones for key distribution and management tasks.

## XII. CONCLUSIONS

In addressing the critical challenge of securing CAN communications in modern vehicles, this study presents and compares the most promising authentication protocols currently under discussion. Identifying essential security requirements as foundational, our analysis evaluates the strengths and weaknesses of these protocols. Additionally, we extend our comparison to include operational criteria, providing a comprehensive understanding of the protocols' specifications and features. In our findings, we observe a prevalent emphasis in existing solutions on fortifying defenses against external attackers, leaving potential vulnerabilities open to internal threats. However, a notable subset of protocols, namely LiBrA-CAN, AuthentiCAN, LCAP, CaCAN, and LinAuth, emerges as capable of providing comprehensive security for message authenticity and system reliability. These protocols, leveraging approaches such as hash chains, asymmetric encryption, and M-MACs, showcase robustness in safeguarding CAN communications. In particular, the efficacy of the hash-chain approach, coupled with asymmetric encryption and M-MAC techniques, stands out as the optimal solution. This comparative analysis underscores the availability of secure and viable authentication solutions within the literature, dispelling any reservations about adoption. Such protocols not only ensure secure communication exchanges but also substantially mitigate potential harm from malicious attacks.

In conclusion, this work contributes valuable insights to the realm of CAN security, advocating for the widespread adoption of authentication mechanisms in in-vehicle communication. By fortifying CAN bus security, our findings aim to motivate and persuade manufacturing industries to embrace these proven and secure authentication solutions, thereby enhancing the overall resilience of automotive systems against cybersecurity threats.

## REFERENCES

- [1] J. Liu, J. Wan, D. Jia, B. Zeng, D. Li, C.-H. Hsu, and H. Chen, "High-efficiency urban traffic management in context-aware computing and 5g communication," vol. 55, no. 1. IEEE, 2017, pp. 34–40.

- [2] J. Contreras-Castillo, S. Zeadally, and J. A. Guerrero-Ibañez, "Internet of vehicles: architecture, protocols, and security," vol. 5, no. 5. IEEE, 2017, pp. 3701–3709.
- [3] Y. Toor, P. Muhlethaler, A. Laouiti, and A. De La Fortelle, "Vehicle ad hoc networks: Applications and related technical issues," *IEEE communications surveys & tutorials*, vol. 10, no. 3, pp. 74–88, 2008.
- [4] H. Hartenstein and L. Laberteaux, "A tutorial survey on vehicular ad hoc networks," *IEEE Communications magazine*, vol. 46, no. 6, pp. 164–171, 2008.
- [5] S. Djahel, R. Doolan, G.-M. Muntean, and J. Murphy, "A communications-oriented perspective on traffic management systems for smart cities: Challenges and innovative approaches," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 1, pp. 125–151, 2014.
- [6] S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, K. Koscher, A. Czeskis, F. Roesner, and T. Kohno, "Comprehensive experimental analyses of automotive attack surfaces," in *20th USENIX security symposium (USENIX Security 11)*, 2011.
- [7] A.-I. Radu and F. D. Garcia, "Leia: A lightweight authentication protocol for can," in *European Symposium on Research in Computer Security*. Springer, 2016, pp. 283–300.
- [8] J. Cook and J. Freudenberg, "Controller area network (can)," in *EECS*, vol. 461, 2007, pp. 1–5.
- [9] S. C. HPL, "Introduction to the controller area network (can)." Texas instruments, 2002, pp. 1–17.
- [10] I. Foster and K. Koscher, "Exploring controller area networks," *login. USENIX Association*, vol. 40, no. 6, 2015.
- [11] R. Buttigieg, M. Farugia, and C. Meli, "Security issues in controller area networks in automobiles," in *2017 18th International Conference on Sciences and Techniques of Automatic Control and Computer Engineering (STA)*. IEEE, 2017, pp. 93–98.
- [12] S. Woo, H. J. Jo, and D. H. Lee, "A practical wireless attack on the connected car and security protocol for in-vehicle can," vol. 16, no. 2, 2015, pp. 993–1006.
- [13] A. Van Herreweghe, D. Singelee, and I. Verbauwhede, "Canauth-a simple, backward compatible broadcast authentication protocol for can bus," in *ECRYPT Workshop on Lightweight Cryptography*, vol. 2011, 2011, p. 20.
- [14] A. Hazem and H. Fahmy, "Lcap-a lightweight can authentication protocol for securing in-vehicle networks," in *10th escar Embedded Security in Cars Conference, Berlin, Germany*, vol. 6, 2012, p. 172.
- [15] T. U. Kang, H. M. Song, S. Jeong, and H. K. Kim, "Automated reverse engineering and attack for can using obd-ii," in *2018 IEEE 88th Vehicular Technology Conference (VTC-Fall)*. IEEE, 2018, pp. 1–7.
- [16] Y. Zhang, B. Ge, X. Li, B. Shi, and B. Li, "Controlling a car through obd injection," in *2016 IEEE 3rd International Conference on Cyber Security and Cloud Computing (CSCloud)*. IEEE, 2016, pp. 26–29.
- [17] 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: 14th International Conference, DIMVA 2017, Bonn, Germany, July 6-7, 2017, Proceedings 14*. Springer, 2017, pp. 185–206.
- [18] G. Bloom, "Weepingcan: A stealthy can bus-off attack," in *Workshop on Automotive and Autonomous Vehicle Security*, 2021.
- [19] K. Iehira, H. Inoue, and K. Ishida, "Spoofing attack using bus-off attacks against a specific ecu of the can bus," in *2018 15th IEEE Annual Consumer Communications & Networking Conference (CCNC)*. IEEE, 2018, pp. 1–4.
- [20] M. Bozdal, M. Samie, S. Aslam, and I. Jennions, "Evaluation of can bus security challenges," *Sensors*, vol. 20, no. 8, p. 2364, 2020.
- [21] C. Miller and C. Valasek, "Remote exploitation of an unaltered passenger vehicle," vol. 2015, no. S 91, 2015.
- [22] S.-F. Lokman, A. T. Othman, and M.-H. Abu-Bakar, "Intrusion detection system for automotive controller area network (can) bus system: a review," *EURASIP Journal on Wireless Communications and Networking*, vol. 2019, pp. 1–17, 2019.
- [23] M. D. Hossain, H. Inoue, H. Ochiai, D. Fall, and Y. Kadobayashi, "Lstm-based intrusion detection system for in-vehicle can bus communications," *IEEE Access*, vol. 8, pp. 185 489–185 502, 2020.
- [24] M. Hanselmann, T. Strauss, K. Dormann, and H. Ulmer, "Canet: An unsupervised intrusion detection system for high dimensional can bus data," *Ieee Access*, vol. 8, pp. 58 194–58 205, 2020.
- [25] K. Tindell, "Can injection: keyless car theft," <https://kentindell.github.io/2023/04/03/can-injection/>, April 2023.
- [26] P. Carsten, T. R. Andel, M. Yampolskiy, and J. T. McDonald, "In-vehicle networks: Attacks, vulnerabilities, and proposed solutions," in *Proceedings of the 10th Annual Cyber and Information Security Research Conference*, 2015, pp. 1–8.
- [27] M. Wolf, A. Weimerskirch, and C. Paar, "Security in automotive bus systems," in *Workshop on Embedded Security in Cars*. Citeseer, 2004, pp. 1–13.
- [28] I. Studnia, V. Nicomette, E. Alata, Y. Deswarte, M. Kaàniche, and Y. Laarouchi, "Survey on security threats and protection mechanisms in embedded automotive networks," in *2013 43rd Annual IEEE/IFIP Conference on Dependable Systems and Networks Workshop (DSN-W)*, 2013, pp. 1–12.
- [29] J. Liu, S. Zhang, W. Sun, and Y. Shi, "In-vehicle network attacks and countermeasures: Challenges and future directions," *IEEE Network*, vol. 31, no. 5, pp. 50–58, 2017.
- [30] 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.
- [31] O. Avatefipour and H. Malik, "State-of-the-art survey on in-vehicle network communication "can-bus" security and vulnerabilities," 2018.
- [32] B. Groza and P.-S. Murvay, "Security solutions for the controller area network: Bringing authentication to in-vehicle networks," vol. 13, no. 1, 2018, pp. 40–47.
- [33] M. Gmiden, M. H. Gmiden, and H. Trabelsi, "Cryptographic and intrusion detection system for automotive can bus: Survey and contributions," in *2019 16th International Multi-Conference on Systems, Signals & Devices (SSD)*, 2019, pp. 158–163.
- [34] M. Bozdal, M. Samie, S. Aslam, and I. Jennions, "Evaluation of can bus security challenges," *Sensors*, vol. 20, no. 8, 2020.
- [35] S. Hartzell, C. Stubel, and T. Bonaci, "Security analysis of an automobile controller area network bus," *IEEE Potentials*, vol. 39, no. 3, pp. 19–24, 2020.
- [36] E. Aliwa, O. Rana, C. Perera, and P. Burnap, "Cyberattacks and countermeasures for in-vehicle networks," *ACM Comput. Surv.*, vol. 54, no. 1, mar 2021.
- [37] H. J. Jo and W. Choi, "A survey of attacks on controller area networks and corresponding countermeasures," IEEE, 2021.
- [38] T. M. Fakhfakh F. and M. M., "Cybersecurity attacks on can bus based vehicles: a review and open challenges," *Library Hi Tech*, vol. 40, no. 5, pp. 1179–1203, 2022.
- [39] M. D. Pesé, J. W. Schauer, J. Li, and K. G. Shin, "S2-can: Sufficiently secure controller area network," in *Annual Computer Security Applications Conference*, 2021, pp. 425–438.
- [40] T. Ziermann, S. Wildermann, and J. Teich, "Can+: A new backward-compatible controller area network (can) protocol with up to 16x higher data rates," in *2009 Design, Automation Test in Europe Conference Exhibition*, 2009, pp. 1088–1093.
- [41] F. Hartwich *et al.*, "Can with flexible data-rate," in *Proc. iCC*. Citeseer, 2012, pp. 1–9.
- [42] C.-L. Wey, C.-H. Hsu, K.-C. Chang, and P.-C. Jui, "Enhancement of controller area network (can) bus arbitration mechanism," in *2013 International Conference on Connected Vehicles and Expo (ICCVE)*. IEEE, 2013, pp. 898–902.
- [43] P.-S. Murvay and B. Groza, "Dos attacks on controller area networks by fault injections from the software layer," in *Proceedings of the 12th International Conference on Availability, Reliability and Security*, 2017, pp. 1–10.
- [44] "Comparing can fd with classical can." [Online]. Available: <https://www.kvaser.com/wp-content/uploads/2016/10/comparing-can-fd-with-classical-can.pdf>
- [45] S. Mavrouniotis and M. Ganley, "Hardware security modules," in *Secure Smart Embedded Devices, Platforms and Applications*. Springer, 2013, pp. 383–405.
- [46] "E-safety vehicle intrusion protected applications (evita)." [Online]. Available: <https://evita-project.org/>
- [47] L. Apvrille, R. El Khayari, O. Henniger, Y. Roudier, H. Schweppe, H. Seudí, B. Weyl, and M. Wolf, "Secure automotive on-board electronics network architecture," in *FISITA 2010 world automotive congress, Budapest, Hungary*, vol. 8, 2010.
- [48] C. Labrado and H. Thapliyal, "Hardware security primitives for vehicles," vol. 8, no. 6. IEEE, 2019, pp. 99–103.
- [49] O. Henniger, A. Ruddle, H. Seudí, B. Weyl, M. Wolf, and T. Wollinger, "Securing vehicular on-board it systems: The evita project," in *VDI/VW Automotive Security Conference*, 2009, p. 41.
- [50] S. Fürst, J. Mössinger, S. Bunzel, T. Weber, F. Kirschke-Biller, P. Heitkampfer, G. Kinkelin, K. Nishikawa, and K. Lange, "Autosar—a worldwide standard is on the road," in *14th International VDI Congress Electronic Systems for Vehicles, Baden-Baden*, vol. 62, 2009, p. 5.



- [51] C. J. McIvor, M. McLoone, and J. V. McCanny, "Hardware elliptic curve cryptographic processor over  $rmgf(p)$ ," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 53, no. 9, pp. 1946–1957, 2006.
- [52] W. Stallings, "The whirlpool secure hash function," *Cryptologia*, vol. 30, no. 1, pp. 55–67, 2006.
- [53] M. J. Dworkin, E. B. Barker, J. R. Nechvatal, J. Foti, L. E. Bassham, E. Roback, and J. F. Dray Jr, "Advanced encryption standard (aes)," 2001.
- [54] "Introduction to j1939." [Online]. Available: [https://cdn.vector.com/cms/content/know-how/\\_application-notes/AN-ION-1-3100\\_Introduction\\_to\\_J1939.pdf](https://cdn.vector.com/cms/content/know-how/_application-notes/AN-ION-1-3100_Introduction_to_J1939.pdf)
- [55] Y. Burakova, B. Hass, L. Millar, and A. Weimerskirch, "Truck hacking: An experimental analysis of the {SAE} j1939 standard," in *10th USENIX Workshop on Offensive Technologies (WOOT 16)*, 2016.
- [56] P.-S. Murvay and B. Groza, "Security shortcomings and countermeasures for the sae j1939 commercial vehicle bus protocol," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 5, pp. 4325–4339, 2018.
- [57] M. Wolf and T. Gendrullis, "Design, implementation, and evaluation of a vehicular hardware security module," in *Information Security and Cryptology-ICISC 2011: 14th International Conference, Seoul, Korea, November 30-December 2, 2011. Revised Selected Papers 14*. Springer, 2012, pp. 302–318.
- [58] H. Schweppe, Y. Roudier, B. Weyl, L. Apvrille, and D. Scheuermann, "Car2x communication: Securing the last meter - a cost-effective approach for ensuring trust in car2x applications using in-vehicle symmetric cryptography," in *2011 IEEE Vehicular Technology Conference (VTC Fall)*, 2011, pp. 1–5.
- [59] C.-W. Lin and A. Sangiovanni-Vincentelli, "Cyber-security for the controller area network (can) communication protocol," in *2012 International Conference on Cyber Security*, 2012, pp. 1–7.
- [60] O. Hartkopp, C. Reuber, and R. Schilling, "Macan - message authenticated can," in *Escar Conference*, 2012.
- [61] R. Kurachi, Y. Matsubara, H. Takada, N. Adachi, Y. Miyashita, and S. Horiata, "Cacan-centralized authentication system in can (controller area network)," in *14th Int. Conf. on Embedded Security in Cars (ESCAR 2014)*, 2014.
- [62] 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)*, 2014, pp. 13–18.
- [63] S. Nürnberger and C. Rossow, "– vatican– vetted, authenticated can bus," in *Cryptographic Hardware and Embedded Systems – CHES 2016*, B. Gierlichs and A. Y. Poschmann, Eds. Springer Berlin Heidelberg, 2016, pp. 106–124.
- [64] B. Groza, S. Murvay, A. V. Herrewewe, and I. Verbauwhede, "Libra-can: Lightweight broadcast authentication for controller area networks," vol. 16, no. 3. Association for Computing Machinery, apr 2017.
- [65] 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*, ser. ACSAC '17. Association for Computing Machinery, 2017, p. 225–237.
- [66] G. Bella, P. Biondi, G. Costantino, and I. Matteucci, "Toucan: A protocol to secure controller area network," in *Proceedings of the ACM Workshop on Automotive Cybersecurity*, ser. AutoSec '19. Association for Computing Machinery, 2019, p. 3–8.
- [67] E. O. Marasco and F. Quaglia, "Authentican: a protocol for improved security over can," in *2020 Fourth World Conference on Smart Trends in Systems, Security and Sustainability (WorldS4)*. IEEE, 2020, pp. 533–538.
- [68] A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. Robshaw, Y. Seurin, and C. Vikkelsoe, "Present: An ultra-lightweight block cipher," in *Cryptographic Hardware and Embedded Systems-CHES 2007: 9th International Workshop, Vienna, Austria, September 10-13, 2007. Proceedings 9*. Springer, 2007, pp. 450–466.
- [69] N. Mouha, B. Mennink, A. Van Herrewewe, D. Watanabe, B. Preneel, and I. Verbauwhede, "Chaskey: an efficient mac algorithm for 32-bit microcontrollers," in *International Conference on Selected Areas in Cryptography*. Springer, 2014, pp. 306–323.
- [70] B. Groza, L. Popa, and P.-S. Murvay, "Tricks—time triggered covert key sharing for controller area networks," *IEEE Access*, vol. 7, pp. 104 294–104 307, 2019.
- [71] H. Wang, D. Forte, M. M. Tehranipoor, and Q. Shi, "Probing attacks on integrated circuits: Challenges and research opportunities," *IEEE Design & Test*, vol. 34, no. 5, pp. 63–71, 2017.
- [72] S. Pinto and N. Santos, "Demystifying arm trustzone: A comprehensive survey," *ACM computing surveys (CSUR)*, vol. 51, no. 6, pp. 1–36, 2019.
- [73] I. T. AG, "Aurixtm family – tc29xtx." [Online]. Available: <https://www.infineon.com/cms/en/product/microcontroller/32-bit-tricore-microcontroller/32-bit-tricore-aurix-tc29xtx/>
- [74] G. Xie, Y. Chen, R. Li, and K. Li, "Hardware cost design optimization for functional safety-critical parallel applications on heterogeneous distributed embedded systems," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 6, pp. 2418–2431, 2017.
- [75] W. Zou, R. Li, W. Wu, and L. Zeng, "Hardware cost and energy consumption optimization for safety-critical applications on heterogeneous distributed embedded systems," in *2018 IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS)*. IEEE, 2018, pp. 1–10.
- [76] G. Xie, W. Ma, H. Peng, R. Li, and K. Li, "Price performance-driven hardware cost optimization under functional safety requirement in large-scale heterogeneous distributed embedded systems," *IEEE Transactions on Industrial Electronics*, vol. 68, no. 5, pp. 4485–4497, 2019.
- [77] Y. Xie, Y. Guo, S. Yang, J. Zhou, and X. Chen, "Security-related hardware cost optimization for can fd-based automotive cyber-physical systems," *Sensors*, vol. 21, no. 20, p. 6807, 2021.
- [78] N. M. Ben Lakhal, O. Nasri, L. Adouane, and J. B. Hadj Slama, "Controller area network reliability: overview of design challenges and safety related perspectives of future transportation systems," *IET Intelligent Transport Systems*, vol. 14, no. 13, pp. 1727–1739, 2020.
- [79] R. I. Davis, S. Kollmann, V. Pollex, and F. Slomka, "Controller area network (can) schedulability analysis with fifo queues," in *2011 23rd Euromicro Conference on Real-Time Systems*. IEEE, 2011, pp. 45–56.
- [80] K. Tindell and A. Burns, "Guaranteeing message latencies on control area network (can)," in *Proceedings of the 1st International CAN Conference*. Citeseer, 1994.
- [81] K. Tindell, A. Burns, and A. J. Wellings, "Calculating controller area network (can) message response times," *Control engineering practice*, vol. 3, no. 8, pp. 1163–1169, 1995.
- [82] Tindell, Hansson, and Wellings, "Analysing real-time communications: controller area network (can)," in *1994 Proceedings Real-Time Systems Symposium*. IEEE, 1994, pp. 259–263.
- [83] P. Thirumavalavasethurayar and T. Ravi, "Implementation of replay attack in controller area network bus using universal verification methodology," in *2021 International Conference on Artificial Intelligence and Smart Systems (ICAIS)*. IEEE, 2021, pp. 1142–1146.
- [84] H. J. Jo and W. Choi, "A survey of attacks on controller area networks and corresponding countermeasures," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 7, pp. 6123–6141, 2021.
- [85] C.-W. Lin and A. Sangiovanni-Vincentelli, "Cyber-security for the controller area network (can) communication protocol," in *2012 International Conference on Cyber Security*. IEEE, 2012, pp. 1–7.
- [86] 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.
- [87] W. Choi, H. J. Jo, S. Woo, J. Y. Chun, J. Park, and D. H. Lee, "Identifying ecus using inimitable characteristics of signals in controller area networks," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 6, pp. 4757–4770, 2018.
- [88] H. Salmani and S. G. Miremadi, "Contribution of controller area networks controllers to masquerade failures," in *11th Pacific Rim International Symposium on Dependable Computing (PRDC'05)*. IEEE, 2005, pp. 5–pp.
- [89] H. Zhang, Y. Shi, J. Wang, and H. Chen, "A new delay-compensation scheme for networked control systems in controller area networks," *IEEE Transactions on Industrial Electronics*, vol. 65, no. 9, pp. 7239–7247, 2018.
- [90] B. Groza and S. Murvay, "Efficient protocols for secure broadcast in controller area networks," *IEEE Transactions on Industrial Informatics*, vol. 9, no. 4, pp. 2034–2042, 2013.
- [91] B. Groza and P.-S. Murvay, "Broadcast authentication in a low speed controller area network," in *E-Business and Telecommunications: International Joint Conference, ICETE 2011, Seville, Spain, July 18-21, 2011, Revised Selected Papers*. Springer, 2012, pp. 330–344.
- [92] G. Cena, I. C. Bertolotti, T. Hu, and A. Valenzano, "Improving compatibility between can fd and legacy can devices," in *2015 IEEE 1st International Forum on Research and Technologies for Society and Industry Leveraging a better tomorrow (RTSI)*. IEEE, 2015, pp. 419–426.

- [93] Lenovo, "What is backward compatible?" [Online]. Available: <https://www.lenovo.com/us/en/glossary/backward-compatible>
- [94] commaai, "opendbc," 2017. [Online]. Available: <https://github.com/commaai/opendbc>
- [95] A. Musuroi, B. Groza, L. Popa, and P.-S. Murvay, "Fast and efficient group key exchange in controller area networks (can)," *IEEE Transactions on Vehicular Technology*, vol. 70, no. 9, pp. 9385–9399, 2021.
- [96] S. Jain and J. Guajardo, "Physical layer group key agreement for automotive controller area networks," in *Cryptographic Hardware and Embedded Systems—CHES 2016: 18th International Conference, Santa Barbara, CA, USA, August 17-19, 2016, Proceedings 18*. Springer, 2016, pp. 85–105.
- [97] Q. Wang, Y. Qian, Z. Lu, Y. Shoukry, and G. Qu, "A delay based plug-in-monitor for intrusion detection in controller area network," in *2018 Asian Hardware Oriented Security and Trust Symposium (AsianHOST)*. IEEE, 2018, pp. 86–91.
- [98] S. U. Sagong, R. Poovendran, and L. Bushnell, "Mitigating vulnerabilities of voltage-based intrusion detection systems in controller area networks," *arXiv preprint arXiv:1907.10783*, 2019.
- [99] T. Hoppe, S. Kiltz, and J. Dittmann, "Applying intrusion detection to automotive it-early insights and remaining challenges," *Journal of Information Assurance and Security (JIAS)*, vol. 4, no. 6, pp. 226–235, 2009.
- [100] P. Vasile, B. Groza, and S. Murvay, "Performance analysis of broadcast authentication protocols on can-fd and flexray," in *Proceedings of the WESS'15: Workshop on Embedded Systems Security*, 2015, pp. 1–8.
- [101] B. Weyl, M. Wolf, F. Zweers, T. Gendrullis, M. S. Idrees, Y. Roudier, H. Schweppe, H. Platzdasch, R. El Khayari, O. Henniger *et al.*, "Secure on-board architecture specification," *Evita Deliverable D*, vol. 3, p. 2, 2010.



## APPENDIX A CRYPTOGRAPHY

This section presents an overview of the cryptographic primitives that are introduced in this paper. We assume we have two entities, Alice and Bob, that want to communicate with each other and an eavesdropper, Eve.

### A. Symmetric Encryption Cryptosystem

Alice and Bob want to communicate secretly, meaning that a possible eavesdropper, Eve, listening to the communication, cannot grasp the meaning of the conversation. Assume that Alice and Bob share a secret  $K$ . We call it *secret key*. In a symmetric encryption cryptosystem, Alice encrypts a secret message  $u$  through a predefined encryption algorithm  $E(u, K)$ . The algorithm takes as input the secret message  $u$ , the secret key  $K$ , and returns in output the encrypted message  $x$ :  $x = E(u, K)$ . Then, Alice transmits  $x$  to Bob, which in turn execute the decryption algorithm  $D(x, K)$ , giving as input the received encrypted message  $x$  and the key  $K$ :  $u = D(x, K)$ . We can then derive that the following relation must hold:  $D_K = E_K^{-1}$ . It is worth noticing that the secrecy of the communication between Alice and Bob comes from the fact that Eve does not know the secret key  $K$ , which is rather shared between Alice and Bob.

### B. Symmetric Authentication

We want to authenticate the communication, ensuring that Bob distinguishes whether a received message comes from Alice or Eve masquerading as Alice. Assume that Alice and Bob share a symmetric key  $k$ . Alice generates a signature  $x$  of the transmitted message in a symmetric authentication mechanism with a signing function  $S(u, k)$ . The function receives as input the message  $u$  and the key  $K$ :  $x = S(u, K)$ . Eve wants Bob to accept a forged message  $u'$  rather than  $u$ . The authentication problem can thus be modeled as a binary problem with a switch  $b = 0$  if the message comes from Alice,  $b = 1$  otherwise. Bob verifies the authenticity of the message thanks to the verification function  $V(\tilde{x}, K)$ , taking as input the received signature  $\tilde{x}$  and the key  $K$ . Hence:  $\hat{b} = V(\tilde{x}, K)$ . If  $\hat{b} = 0$ , then Bob accepts the message, as this is coming from the legitimate transmitter; if instead  $\hat{b} = 1$ , the message is rejected.

### C. Asymmetric Encryption Cryptosystem

In an asymmetric encryption cryptosystem, there is no shared key between Alice and Bob. Rather, each entity possesses a couple of keys: a *private key*  $K$  and a *public key*  $K' = f(K)$ . While the former is private and only known by the owner, the latter is known to anybody. There is a special relationship between the two keys, so if Alice encrypts a message using Bob's public key, only Bob can decrypt it as he is the only one who possesses the corresponding private key. This system ensures that anyone can encrypt a message, but only the legitimate receiver can remove the encryption. The security of an asymmetric encryption cryptosystem is based on the following assumptions: (i) it is computationally hard

to derive  $K$  from  $K'$ ; (ii) it is computationally hard to derive the message  $u$  given  $K'$  and the encrypted message  $x$ ; (iii) it is computationally hard to derive  $K$  from  $u$  and  $x$ .

### D. Cryptographic Hash Functions & Hash Chains

In general, a *hash function*  $h : \mathcal{X} \rightarrow \mathcal{Y}$  maps an arbitrarily long input  $x \in \mathcal{X}$  into a fixed-length output  $y \in \mathcal{Y}$ , called as *hash value*. The hash value is unique to the input data and should be uniform in  $\mathcal{Y}$ . A *cryptographic hash function* is a special class of hash functions for which the following properties must hold:

- 1) Pre-Image Resistance – Given  $y \in \mathcal{Y}$  it must be computationally difficult to retrieve  $x$  such that  $h(x) = y$ .
- 2) Second Pre-image Resistance – Given  $x_1 \in \mathcal{X}$ , it must be computationally difficult to find  $x_2 \in \mathcal{X}, x_2 \neq x_1$  such that  $h(x_2) = h(x_1)$ .
- 3) Strong Collision Resistance – It must be computationally hard to find any pair  $(x_1, x_2) \in \mathcal{X}^2, x_1 \neq x_2$  such that  $h(x_1) = h(x_2)$ .

A *hash chain* is a collection of hash values  $\{y_0, \dots, y_n\}$  where the following relation holds:  $y_0 = h(x), y_i = h(y_{i-1}), i = 1, \dots, n$ , and  $h(\cdot)$  being a cryptographic hash function.