# Secure Authentication in the Presence of Malicious Messages and Packet Reorders: Study on CAN Bus

1st Sofia Maragkou
*Institute of Computer Technology*
*Vienna University of Technology*
Vienna, Austria
sofia.maragkou@tuwien.ac.at

2rd Miltos Grammatikakis
*ECE Dept.*
*Hellenic Mediterranean University*
Heraklion, Greece
mdgramma@cs.hmu.gr

3nd Nikos Papatheodorou
*ECE Dept.*
*Hellenic Mediterranean University*
Heraklion, Greece
ntpapatheodorou@cs.hmu.gr

4rd Axel Jantsch
*Institute of Computer Technology*
*Vienna University of Technology*
Vienna, Austria
axel.jantsch@tuwien.ac.at

2025 IEEE 15th Annual Computing and Communication Workshop and Conference (CCWC) | 979-8-3315-0769-5/25/$31.00 ©2025 IEEE | DOI: 10.1109/CCWC62904.2025.10903953

***Abstract*—Message authentication is fundamental for securing modern automotive networks. Our work focuses on integrating buffering in existing authentication protocols to sustain the presence of malicious or corrupt messages, and arbitrary packet swaps in the in-vehicle network. The proposed extension applies to the popular vatiCAN protocol, and other CAN bus authentication protocols, which use separate messages for transferring packet information and authentication data. The proposed extension uses one or more, independent Finite State Machines (FSMs) at each receiver node to temporarily store and subsequently validate message pairs, i.e., a legitimate data packet L with its hashed-based message authentication code (HMAC) packet H. The proposed methodology is evaluated experimentally on a Raspberry Pi-based Electronic Control Unit (ECU) with CAN interfaces. We examine key design parameters, such as the LH swap rate, the malicious rate, and queue configuration options, such as the queue size and flush policy. Results show that the protocol extension improves authentication. When the queue size is below 5, the LH swap rate is up to 50%, and 50% of malicious packets are introduced, the validated packet rate is low (5%). However, if the queue size exceeds 20, the verified packet rate reaches 100%, regardless of other parameters. The increased queue size has a minimal effect on latency, which increases by a few ms on average, and on false positives, which remain below $10^{-9}$. Statistical models help evaluate queue size bounds for worst-case scenarios, strengthening our experimental findings.**

***Index Terms*—automotive security, CAN bus, hash-based authentication, in-vehicle networks, reliability, FSM**

## I. Introduction

Safety and security are interrelated system properties in many application domains. Ensuring just the safety of automotive vehicles is no longer sufficient. Security has become equally important, as exhibited in numerous examples. In the attack known as "The Jeep Hack" [1], the car engine was remotely controlled by first exploiting a software vulnerability in the infotainment system. Similarly, in [2], the authors have shown how to execute system commands using the radio, the Bluetooth stack, or the telematics unit, thus remotely affecting physical components of the vehicle through CAN message injection, message spoofing, and replay attacks. These attacks show that vehicle communication vehicle subsystems are particularly vulnerable to attacks.

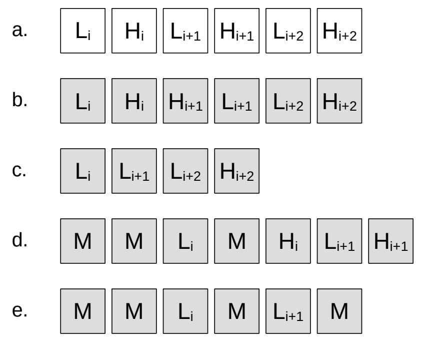

Fig. 1. In the above images the order of received packets at a network node is shown. Scenario a: Packets are received according to the specifications, i.e., each legitimate packet L (carrying data) is followed by its authentication H (carrying an HMAC). Scenario b: Packets are received in the wrong order, i.e., certain (L, H) pairs have been swapped. Scenario c: Not all packets are received. Scenario d: Malicious or corrupt packets are received among legitimate packets and their authentication. Scenario e: Malicious or corrupt packets are received, and legitimate packets and/or authentications are lost.

Since then, several holistic security solutions have focused on strengthening authenticated broadcast communication in-vehicle networks using lightweight proprietary and open source cryptographic protocols [3]–[5]. However, hazards related to a) the presence of malicious or corrupt messages or b) packet reordering due to automotive network reliability issues (e.g., poor channel conditions) can cause complete failure of these security protocols (as seen in [6], [7]). In particular, if a malicious or corrupt packet appears, the subsequent pairing of an incoming packet L and an authentication packet H carrying an HMAC is broken. Moreover, thereafter, all remaining packets are erroneously authenticated.

Figure 1 presents different cases of packets received at a network node. When packets are broadcast over an in-vehicle bus, such as a CAN Bus, case (a) presents queuing at the receiver in the simplest case of paired (L, H) packets. Cases (b)-(e) present different scenarios due to a) message reorders caused by the introduction of malicious or corrupt packets, or b) the loss of packets caused by unreliable communication, deliberately shutting down an interface in case of intrusion, bus off, etc. In this paper, we consider scenarios (b) and (d), but not (c) and (e) where packets are lost.

Note that with existing security protocols, when a malicious packet is received, the receiver node has no knowledge of it. Thus, it behaves normally, trying to authenticate the packet as a legitimate one, and fails. This can become a major issue since the relative order of the legitimate packets and their corresponding authentication packets is broken. Given that the receiver cannot distinguish between legitimate (L) and hashed packets (H), if another possibly malicious or corrupt packet intervenes, the receiver tries to authenticate two non-paired packets and likely fails. Hence, depending on the number of packet reorders and malicious or corrupt packets at the receiver, the efficiency of the authentication mechanism is affected and may drop to zero. Depending on the number of malicious packets, legitimate packets may never be authenticated, or in the worst case, the system may become unavailable (denial-of-service).

This paper contributes to the following solution.

- A novel, robust security protocol that allows the receiver to tackle message reordering and injection of malicious or corrupt packets. The protocol improves the robustness of hash-based authentication protocols that use separate packets for data and authentication. To the extent of our knowledge, no similar techniques have been documented and evaluated in the literature.

We validate the proposed protocol extension in two ways:

- We present an experimental case study on the CAN Bus, using an open distributed embedded platform consisting of Raspberry Pi 3, Model B+ nodes connected to CAN using dual Canberry SPI-to-CAN interfaces.
- We also provide a worst-case model using statistical arguments. More specifically, if malicious/corrupt packets are generated by independent random processes, then the sequence of precisely $k$ consecutive malicious packets (called a loss run) in a run of $N$ packets corresponds to a Bernoulli trial of size $N$ and loss probability $q$ (probability of malicious packet). The loss run represents a number of malicious packets that arrive together.

Both experimental and statistical results demonstrate the high performance and reliability achieved by our security protocol extension, with small queues of size up to 20 packets.

In Section II, the related work is presented. Then, in Section III, we describe our methodology, explaining how an FSM at the receiver node manages a queue to improve authentication performance and reliability, by handling reordering and malicious or corrupt messages. The experimental framework is examined in Section IV and the results are presented in section V. We conclude our paper with future work in Section VI.

## II. Related work

The Controller Area Network (CAN bus) is a bus protocol (ISO standard 11898) widely used for in-vehicle communication. It supports real-time data exchange between different nodes, such as sensors, actuators, and electronic control units (ECUs). The baud rate of the protocol can reach up to 1Mb/s and the data frame includes 8-byte data with an 11- or 29-bit ID. The nodes broadcast the messages asynchronously, and the messages are prioritized for transmission based on their ID (PID field). The lower the PID, the higher the priority.

CAN bus protocol does not support security features. This leads to security exploits, as explained in [8]. The result of the assessment as well as the vulnerabilities described in [9], regarding possible security attacks on autonomous vehicles and their countermeasures, are summarized in Table I.

TABLE I
The vulnerabilities of the standard CAN protocol.

| Properties | Attacks | Explanations | Countermeasures |
|---|---|---|---|
| Confidentiality | Eavesdropping | CAN packets are plaintexts. | Encryption |
| Integrity | Man-in-the-middle, replay, spoofing, data alteration | Messages are broadcasted. | Authentication |
| Availability | Denial of service (DoS) | Priority is based on PID | Anomaly detection, firewall |
| Device Authenticity | Spoofing, replication, Sybil attack | Devices are not authenticated | Device fingerprints |

To cope with different vulnerabilities described in Table I, a variety of authentication and encryption protocols for CAN bus have been developed. In our study, we concentrate on a class of security protocols that use separate packets for data and authentication. At the expense of extra packet delay, these protocols increase both security and reliability since alternative solutions use fields in the same packet to carry authentication data, e.g., bits in the data byte field or the cyclic redundancy check (CRC) field. Thus, we focus next on the three popular representatives of this class: vatiCAN, vecure, and VulCAN.

vatiCAN is a backwards-compatible add-on to the basic CAN protocol [3]. To reduce the cryptographic overhead for legacy devices, vatiCAN is used only for critical messages. Authentication messages are sent separately with a special PID, keeping backward compatibility with CAN bus. For message authentication, lightweight symmetric cryptography with a hash-based authentication code (HMAC) is used based on its compliance with real-time and the resource-constrained nature of the ECUs. During normal operation, vatiCAN tackles replay and spoofing attacks. For the replay attack, a random nonce is added to each hashed-based message authentication code (HMAC) computation. For the nonce, a counter is used at each sender. This counter is incremented after each transmission. If the sender uses a unique set of PIDs, spoofed messages sent by a compromised device can be detected by comparing them with the original PIDs. Notice that to deal with the effect of message loss during synchronization, a nonce synchronization channel has been introduced using timing-based covert channels [4]. The vatiCAN authentication scheme can benefit from Vecure and VulCAN secure primitives.

Vecure uses trust levels to group ECUs [5]. The high-trust group shares a symmetric secret key. Authentication frames contain a 2-byte counter, a 1-byte node ID, a 1-byte marker, and a 4-byte authentication code. This frame is sent like vatiCAN in a second packet. The frame is created either offline using heavyweight SHA3 computation that involves counter, ID, session number, and overflow (device authentication), or using a lightweight hash on data (data authentication).

VulCAN utilizes secure system primitives at the software and hardware level, including a trusted computing base inside which message authentication code (MAC) is generated [10]. Like vatiCAN and Vecure, VulCAN authentication takes a few ms. Its two main features are fast, secure authentication, and software component isolation. In regard to security, to keep the protocol compatible with legacy systems, the authentication data, and the message packets are transmitted separately. The MAC is computed over a 128-bit key, a nonce that protects against replay attacks, and the CAN identifier, in the case the key is shared among different IDs. To protect from replay attacks, the nonce value is never repeated for the same key and data, thus the initialization of the nonce is vital. For this reason, session keys are generated, and the nonce is initialized. In the case of packet loss, nonce is out-of-sync and authentication will fail. Depending on the case, the nonce can be sent in the message payload.

### A. Comparisons with Related Work

Fault tolerance related to packet reordering and malicious or corrupt message injection is important in critical systems, such as in-vehicle automotive networks. Packet reordering, along with time synchronization, have been used successfully for designing a covert data channel for encoding forward error correction when vatiCAN experiences packet loss [4], since using re-transmissions is expensive. Therefore, packet reordering can actually occur on CAN.

Although popular CAN-based protocols, such as vatiCAN [3], Vecure [5], or VulCAN [10], have considered packet loss and/or replay (duplication) attacks as a threat model, a packet reordering threat model has not been considered in the context of automotive in-vehicle networks. However, since the above CAN-based authentication protocols use pairs of packets for authentication, even a simple packet reorder (or a malicious or corrupt message) breaking the pair relationship can cause catastrophic consequences: all following packets to be erroneously authenticated.

Packet reordering attacks can be implemented by a malicious CAN driver present on the sender node or an external malicious ECU node that acts as a man-in-the-middle. The proposed security extension provides a way to restore pairing at the receiver by utilizing a queue that stores unauthenticated packets for future comparisons. This method improves the attack resilience of hash-based authentication protocols that use separate packets for data and authentication information.

Both experimental results and theoretical statistical models are developed to demonstrate good system performance and reliability with limited queue size for the proposed extension that addresses packet reordering threats.

## III. Methodology

This section is focused on handling authentication at the receiver in the presence of packet reordering, and malicious or corrupt messages. Message corruption can occur because of channel issues or malicious actions. Confidentiality of the transferred data is beyond our scope. However, the treatment of confidentiality could impact authentication as well, depending on the scheme, e.g., encrypt-then-MAC, or MAC-then-encrypt.

The sender, for each one of the data packets (also referred to as legitimate packet) (L), sends an additional authentication packet H, always in this order. The authentication packet contains the hash value of the legitimate packet. This value is used to verify the integrity of the legitimate packet. The sender node is not known to the receiver. The only related packet information apart from the data bytes for L (or authentication bytes for H) is the packet priority (PID). The lower the PID, the higher the priority of the packet. This helps in avoiding collisions when nodes broadcast packets to the CAN bus. The normal sequence is $L_1, H_1, L_2, H_2, L_3, H_3$, which allows the authentication process to proceed without any issues.

We consider that the packets are received in the correct order per node, though, this does not exclude the possibility that reordered packets from different nodes can exist. Reordering is caused by inconsistent merging of message flows at the receiver due to limited knowledge. This effect can create a sequence at the receiver (per PID) as the following: $L_1, L_2, H_1, H_2, L_3, H_3$ where $L_i$ is the legitimate data packet from node $i$ and $H_i$ is the corresponding authentication packet from node $i$. There are also several other possible sequences, such as $L_1, L_2, H_2, H_1, L_3, H_3$

The receiver has no previous knowledge of malicious or legitimate packets. Hence, ignoring the case of rare false positives (examined later in this section), the receiver can only assume that all unauthenticated packets are malicious or corrupt. To restore pairing a centralized queue stores unauthenticated packets at the receiver for future comparisons. The behavior of the receiver FSM is analyzed next.

### A. Finite State Machine at the Receiver

To handle authentication with packet reordering and malicious or corrupt messages we resort to queuing. More specifically, instead of using a central queue that stores all pending authentications, we distribute these messages to different statically allocated queues and define one for each different message PID; while CAN base frames support an 11-bit identifier, extended frames support 29-bit identifiers.

The queue in figure 1 (a) represents a valid queue state at the receiver when there is no packet reordering and no malicious or corrupt message. Moreover, diagram 2 describes the FSM at the receiver node. When a new packet is received, the receiver has no knowledge of the packet type (L, H) and pairing, and so it searches the queue until a successful authentication packet is found. During this search, the algorithm examines all possible packets as candidates of an (L, H) pair, By definition, all these packets have the same PID priority.

This process starts with the packets closer to the most recently received packet. This heuristic is expected to increase the possibility of finding matching authentications since packets are always received in the right order per sending node.

In case of successful authentication, the (L, H) packet pair is removed from the queue. Alternatively, in the case of a failed search, the FSM waits for a newly received packet to start a new round of packet authentication comparisons.

If during the process of receiving messages, the queue becomes full, a percentage of the queue packets are deleted starting from the older packets, according to the selected *flush policy*. This happens because, with our heuristic algorithm, packets staying very long in the queue are considered by the receiver as malicious or corrupt, even though there could only be reorders in the queue. Since most messages are periodic, packet re-transmission is not always required by the specific application.

Next, we proceed to examine figure 2 closely. The behavior of the receiver, as well as the efficiency of the proposed methodology, depends on two parameters: the size of each queue (we define one queue for each active $PID$) and their corresponding flush policy.

Let the size of each queue at the receiver be $Q_{max}$. Without loss of generality, let $Q_{max} = 2n + 1$, where $n > 1$ is an integer. For each $PID$, the receiver node uses a finite state machine (FSM) similar to the one shown in figure 2. The states of the FSM correspond to queue sizes $S_0,\ S_1,\ ...,\ S_{Q_{max}}$ based on the number of received packets. We define $S_i$ as the state that corresponds to queue size $i$, where

$$S_i, 0 \leq i \leq 2n + 1 \tag{1}$$

Using separate queues, one for each *PID*, directly aligns messages for subsequent authentication comparisons. Hence, considering authentication only within packets of the same priority minimizes the number of comparisons between unmatched packets. The final queue size created is *number of PIDs* $*Q_{max}$. Practically, since the number of PIDs in different vehicles usually ranges from 6 to 14 or so, and the expected queue size is relatively small (see section IV) the memory requirements at the receiver are relatively small (few KBytes). Moreover, only lightweight processing is needed on each state of the FSM; more specifically, $O(1)$ comparisons are necessary to compare each newly received message with packets already in the queue. For every new packet entry in the queue, the receiver attempts to authenticate it with all existing packets in the queue, starting with the packets received last.

For the states, $S_0, S_1$, the receiver does not attempt to perform any authentication since the number of packets is not sufficient. The authentication starts at first in state $S_2$, where there are 2 packets in the queue. Upon successful authentication, the packets are forwarded to be processed, and they are flushed from the queue, returning to State $S_0$. Similarly, according to diagram 2, it is simple to follow the FSM behavior for larger queue sizes, due to the symmetry implied by taking $Q_{max} = 2n + 1$.

## B. Security analysis and False Positives

A consecutive successful run of two elements, i.e., $L_i,\ H_i$ (corresponding to the correct delivery of a packet and its authentication) is handled correctly without false positives. However, when a corrupt or malicious packet is received after $L_i$, but before $H_i$, it may create a false positive.

Assuming a random attack with corrupt or malicious messages, a single false positive can occur with probability $2^{-N}$, where $N$ is the size of HMAC, i.e., for CAN bus, $N = 64$ bits, since we use 8 data bytes for the authentication. Therefore, with a number of $t$ trials, the probability of at least one false positive is $1-[1-2^{-N}]^t$. Note that this probability remains less than $10^{-9}$, for billions of trials $t$. Therefore, in the context of automotive safety, this error is usually considered acceptable.

## C. Key management

Key management involves key distribution, renewal, and storage, vital processes to CAN authentication. Session keys, used in hash functions, are renewed periodically by a dedicated node. This node can generate and share session keys regularly, protecting their distribution against replay attacks using parameters, such as epoch counters. Although key management is important, it is orthogonal to our work.

## D. Queue Size and Flush Policy Considerations

The efficiency of the proposed authentication partially depends on the queue size $Q_{max} \geq 3$ and the flush policy.

Notice that if the queue is small, it may overflow, thus breaking possible future authentications. This could happen if there is a “sufficiently long” sequence of malicious or corrupt messages or consecutive reorders of many normal packets (e.g., $L_i,\ L_j$) or their authentications ($H_i,\ H_j$). Moreover, if the queue is large, the number of authentication comparisons with previously received packets increases, rendering the inspection of messages in real time impossible. This case also increases the number of false positives.

A key strategy that influences the efficiency of the receiver FSM in authenticating packets is queue overflow management. A progressive overflow management scheme could be to delete many old messages “as soon as possible” (or “as soon as seen necessary”) with the prospect of making space to store and match new upcoming packets. On the other hand, with a more conservative drop-out policy, malicious or corrupt packets may stay in the queue for longer, enabling further authentication comparisons with new upcoming packets. Considering the implications of these two extreme policies, we decide to follow a simple and general, threshold-based, flush policy, as a trade-off between successful authentications (validations) and memory efficiency. More specifically, as shown in diagram 2, the queue is flushed to become completely (or partly) empty,

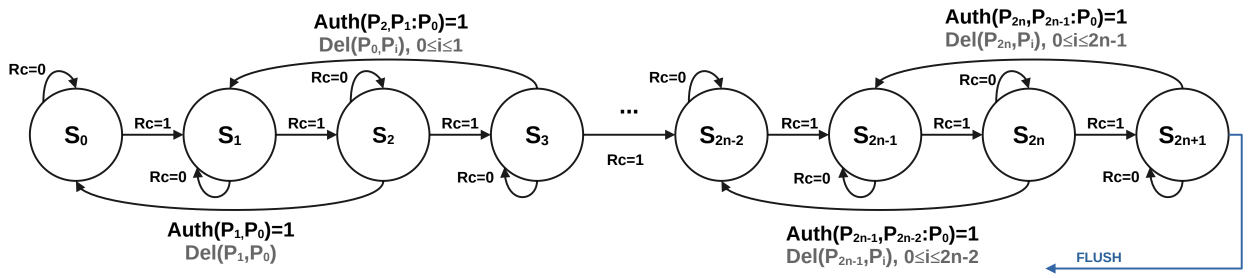

Fig. 2. The finite state machine of the receiver. The condition $Rc = 1$ indicates new packet arrivals, while $Rc = 0$ indicates the absence of a new packet. Besides storing new packets upon arrival, the FSM uses two functions, namely $Auth$ and $Del$. $Auth(P_{2n}, P_{2n-1} : P_0)$ represents authentication comparisons of packet $P_{2n}$ with every packet from $P_{2n-1}$ to $P_0$. The $Del$ process represents the dequeue of a message pair that has authenticated successfully. The $FLUSH$ function of the queue brings the FSM to an earlier state by removing several packets, depending on the queue flush policy selected.

when it becomes full, i.e., when control arrives at the last state of the queue ($Q_{max}$). Notice that this occurs independently for each queue that corresponds to a specific PID.

## E. Statistical Models: Predicting the Queue Size

When considering malicious messages as random processes, the event that “a malicious or corrupt packet arrives” follows a Bernoulli trial process of size $N$ with a success probability of $p$, where $p$ is the probability of a legitimate packet, and $q = 1 - p$ is the loss probability corresponding to a malicious, or corrupt packet. Within this context, we consider a sequence of $k$ consecutive malicious packets (called a loss run) in a run of $N$ trials, preceded and succeeded by zero or more successes. The length of a loss run is a useful parameter since it represents the number of malicious packets that can arrive together. Thus, if the queue size is selected to match the expected value of the loss run length, our FSM will likely not drop packets without pairing them first with legitimate packets for the same $PID$.

If malicious packets arrive at a very high rate compared to legitimate packets $(q >> p)$, a different type of denial-of-service will occur, causing the queue to continuously overflow. Both authentication-based and other metrics, such as the cumulative sum of message frequency counters, can be used to detect this anomaly. Now, focusing on the case where legitimate packets are more than malicious ones $(p >> q)$, figure 3 shows the number of Bernoulli trials $N$ versus the loss probability ($q = 1 - p$), for various numbers of loss runs. From the figure, observe that, if the rate of malicious or corrupt packets falls to 10%, then the number of packets for a loss run of size $k = 30$ increases exponentially to $\approx 10^{30}$. Hence, due to the extremely large number of messages, overflows are not expected to occur with a queue of size 30, if the rate of malicious packets is below 10%. Note that useful asymptotics are provided in [11], and closed-form equations for the case $p = q$ are available in [12] [13].

## IV. Experimental Set up

To evaluate the performance of the security extension, we use the embedded platform in Figure 4. This proof-of-concept platform represents a simplified CAN-based ecosystem with two Raspberry Pi 3B+ CAN nodes: message sender and receiver. Both nodes operate on 2019-04-08-Raspbian, with

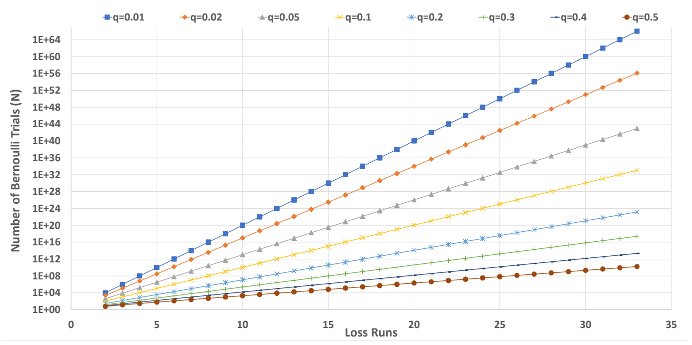

Fig. 3. Bernoulli trials $N$ vs. loss runs (k) for various loss probabilities ($q$). The number of Bernoulli trials for a given loss run increases exponentially when the success probability increases linearly.

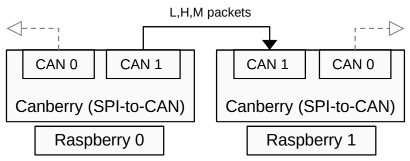

Fig. 4. Schematics of the embedded platform used for experimentation. ECU nodes are based on Raspberry Pi 3B+ with SPI-to-CAN interfaces. The gray arrows indicate other CAN connections.

Linux kernel 4.9 and preempt_rt patch. Using the Industrial-Berry’s Canberry Dual v2.1 SPI-to-CAN shield, the platform can be extended to interface with 4 CAN networks. We configure the CAN rate at $500Kb/sec$, and the transmit queue length of the CAN network interface to a large value ($1M$); this determines the maximum number of packets allowed per transmit queue.

To simulate how the FSM at the receiver handles incoming packets, the sender node generates a sequence of packets based on four parameters, a) a number of legitimate packets ($leg$), b) a number of malicious packets ($mp$), c) the LH packet swap rate ($LH$), and d) the malicious rate ($mal\_rate$). In the experiments, for each legitimate packet, a subsequent authentication packet is transmitted. This packet contains the hash value of the original packet. According to the vatiCAN protocol, the hash value is computed using the SHA3-256 algorithm (in 256 bits) and truncated to 64 bits. Table II presents the specific

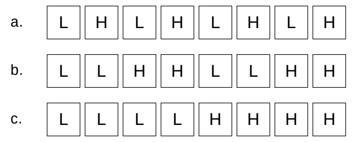

Fig. 5. Typical packet distributions for various LH swap rates. a) Queue for $LH = 0$. b) Queue for $LH = 0.5$; the LH sequences are not so symmetric, i.e., 3 (or more) consecutive L are possible. c) Queue for $LH = 1$.

TABLE II
SENDER PARAMETERS

| Parameter | Investigated values |
|---|---|
| Legitimate packets (leg) | $\{2000, 5000\}$ |
| Malicious packets (mp) | $\{0, 0.5, 1, 2\} * leg$ |
| LH swap rate (LH) | $\{0, 0.5, 1\}$ |
| Malicious rate (MAL) | as defined in Eq. 2 |

parameter values used during the experiment.

The experiments are conducted for 2000 and 5000 legitimate packets. The number of malicious packets is either $0, 1000, 2000, 4000$, for 2000 legitimate packets, or $0, 2500, 5000, 10000$, for 5000 legitimate packets.

The parameter LH swap rate defines the frequency of non-consecutive (L, H) pairs. Higher values result in greater densities of L packets since the H packets remain at the end of the sequence. Figure 5 demonstrates how different LH swap rates impact the packet queue. Higher LH rates increase the probability that packets stay at the receiver's waiting queue for longer, potentially leading to queue overflow.

The malicious rate ($mal\_rate$) defines the distribution of malicious packets, as calculated in equation 2.

$$MAL = mp * (2 * leg + mp)^{-1} \tag{2}$$

where $leg$ is the number of legitimate packets, and $mp$ is the number of malicious packets (see Table II for possible values).

The behavior of the *receiver* is determined by two parameters: a) the queue size and b) the flush policy. The queue size defines the memory space for storing packets not yet authenticated, and the flush policy defines the number of packets deleted in case of queue overflow. The experimental values for these parameters are presented in Table III. Notice that if the flush policy $FL = 0.5$, then the first half of the queue is deleted during a queue overflow. If $FL = 1$, then the whole queue is deleted.

We focus on the following experimental system metrics:

1) The percentage of the validated packets for a given LH swap rate, and malicious rate.
2) The percentage of the validated packets for a given LH swap rate, malicious rate, and flush policy.
3) The average queue length for a given queue size, LH swap rate, and malicious rate.
4) The latency for a given LH swap, and malicious rate.

## V. RESULTS

For the specific experimental configuration, there was no packet loss at the CAN interface. Thus, the receiver node receives all the packets sent from the sender node.

TABLE III
RECEIVER PARAMETERS

| Parameter | Experimental values |
|---|---|
| Queue size | $\{3, 4, 5, 6, 8, 9, 10, 15, 20, 30\}$ |
| Flush policy (FL) | $\{0.5, 1\}$ |

Experimental data collected on the receiver node, includes the number of received packets ($rcv\_p$), the number of validated packets ($val\_p$), the number of deleted packets ($del\_p$), and the number of packets remaining in the queue ($rem\_p$) used for validation. After performing several sanity tests, we analyzed three fundamental system performance metrics. First, the average queue length is calculated using Eq. 3. Second, the average number of authentication trials is computed using Eq. 4. Finally, the total latency is obtained using Eq. 5.

$$avg\_queue\_length = \frac{\sum_{n=1}^{rcv\_p} queue\_length_n}{rcv\_p} \tag{3}$$

where $queue\_length_n$ is the length of the queue upon arrival of packet $n$, and $rcv\_p$ is the total number of received packets.

$$avg\_auth\_trials = \frac{total\_trials}{rcv\_p} \tag{4}$$

where $total\_trials$ is the total number of authentication comparisons that took place in total for all received packets, and $rcv\_p$ is the number of received packets.

$$auth\_latency = \frac{\sum_{n=1}^{val\_p} auth\_delay_n}{val\_p} \tag{5}$$

where $auth\_delay_n$ is the time from the moment the packet $n$ is received until the moment the packet is successfully authenticated, and $val\_p$ is the number of successfully authenticated legitimate packets.

The total latency, average queue length, and number of validated packets, are significantly affected by queue size, flush policy, LH swap rate, and malicious rate.

Upon analyzing figure 6, it becomes evident that queue sizes greater than 20 result in a 100% validated packet rate. Moreover, for queue sizes below 20, the percentage of validated packets increases significantly with the queue size, depending on the LH swap and malicious rates. Lower values of LH swap rate and malicious rate lead to a faster increase in the percentage of validated packets. For instance, when $LH = 0$ and $MAL = 0$, a queue size of 3 items results in a 100% validated packet rate, while $LH = 0.5$ and $MAL = 0.5$ require a queue size of 20 for the same result.

Additionally, the flush policy also affects the validated packets. As illustrated in figure 7, $LH = 0.5$ and $MAL = \{0.2, 0.33\}$ result in lower percentages of validated packets for $FL = 1$ compared to $FL = 0.5$, regardless of the queue size.

The average queue length shown in figure 8 remains relatively low when $MAL = 0$. As the LH swap rate and malicious rate increase, the average queue length increases linearly, justifying the need for a larger queue size (near 25).

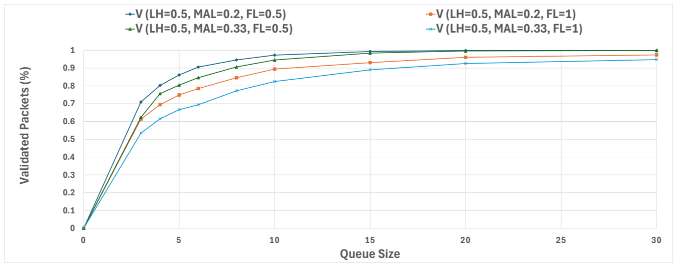

Fig. 6. The percentage of validated packets vs. the queue size for different swap rates ($LH$), and malicious rates ($MAL$); the flush policy $FL = 0.5$.

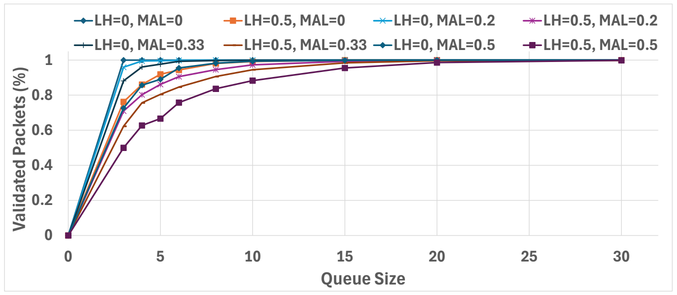

Fig. 7. The percentage of validated packets vs. the queue size for swap rate LH=0.5, malicious rates MAL={0.2,0.33}, and flush policies FL={0.5,1}

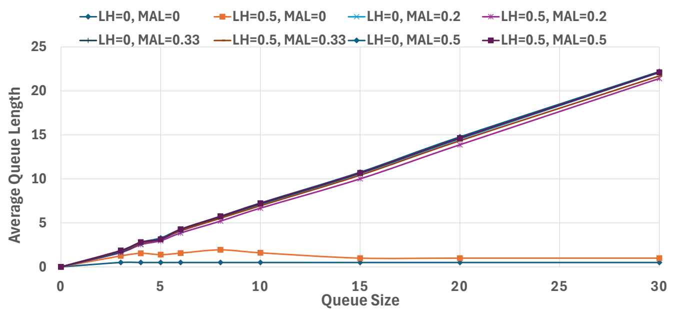

Fig. 8. The average queue length vs. the queue size for different swap rates ($LH$), and malicious rates ($MAL$), given that the flush policy $FL = 0.5$.

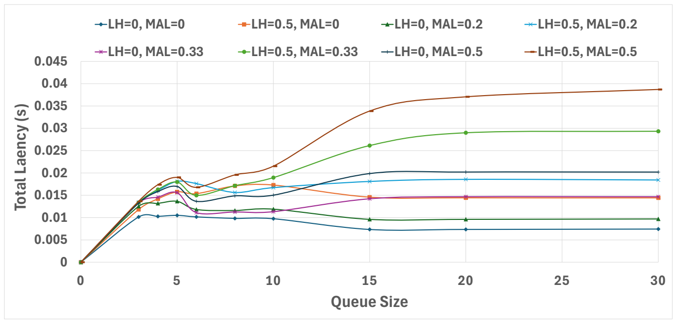

Fig. 9. The total latency of authentication for validated packets based on the queue size for different LH swap rates ($LH$), and the malicious rates ($MAL$).

In figure 9, we examine the total latency of validated packets for $LH = 0.5$. For $LH = 1$, the total latency cannot be calculated since none of the packets is validated. We observe that the total latency increases when the LH swap rate ($LH$), and the malicious rate ($MAL$) increase, although the level of increase is non-linear. In fact, for all $LH$ and $MAL$ rates, the latency increase for a queue size of 20 (versus a queue size of 5) is very small, only a few ms on average. Improvements are marginal since they correspond to statistically insignificant events, as demonstrated in section III-E.

Another important remark is that for queue sizes above 20 packets, where almost all packets are validated (cf., figures 6 and 7), the effect of increasing the malicious rate on the total latency is more profound than increasing the LH rate.

As a practical example, assuming a queue size of 20, and comparing the use case of $LH = 0.5$ and $MAL = 0.5$ with that of $LH = MAL = 0$, we discover that the former case has a $\approx$5 times larger latency (37ms vs 7.3ms). This extra latency of $\approx$30ms extends braking distance by 0.83m on a car speeding at 100 km/h.

Finally, note that by increasing the number of PIDs and providing extra buffering, it is possible to reduce the number of queue entry comparisons. However, although different PIDs (associated with multiple FSMs) may be activated to reduce comparisons, only one of the FSMs would be active each time at each network node, the one receiving the new incoming packet. Hence, our proposed scheme always uses a constant number of comparisons for each received packet.

## VI. Future Work

Several interesting open issues relate to extensions of the current methodology.

- Is it feasible to authenticate multiple messages together with a single tree-based HMAC? Examine trade-offs between performance and security.
- In addition to packet reordering, and malicious or corrupt messages, consider the case of lost messages. For this case, is it possible to apply encoding techniques, such as Reed-Solomon or LPDC error-correcting techniques, in real time? Would these schemes require restructuring to avoid re-computations?

It is also interesting to consider another approach based on multiple hash chains and evaluate related performance, energy, and storage trade-offs.

## References

[1] C. V. Charlie Miller. 2015. Access date: 10/3/2023. [Online]. Available: https://illmatics.com/carhacking.html

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

[3] S. Nürnberger and C. Rossow, "–vatican–vetted, authenticated can bus," in *Cryptographic Hardware and Embedded Systems–CHES 2016: 18th International Conference, Santa Barbara, CA, USA, August 17-19, 2016, Proceedings 18*. Springer, 2016, pp. 106–124.

[4] S. Vanderhallen, J. Van Bulck, F. Piessens, and J. T. Mühlberg, "Robust authentication for automotive control networks through covert channels," *Computer Networks*, vol. 193, p. 108079, 2021.

[5] Q. Wang and S. Sawhney, "Vecure: A practical security framework to protect the can bus of vehicles," in *Proceedings of Int. Conf. on the Internet of Things (IOT)*, ser. IOT '14. Cambridge, MA, USA: Association for Computing Machinery, 2014, p. 13–18.

[6] S. Vanderhallen, J. V. Bulck, F. Piessens, and J. T. M"uhlberg, "Robust authentication for automotive control networks through covert channels," *in Computer Networks*, vol. 193, July 2021.

[7] M. Schulze, P. Werner, G. Lukas, and J. Kaiser, "Afp-an adaptive fragmentation protocol supporting large datagram transmissions." *J. Commun.*, vol. 6, no. 3, pp. 240–248, 2011.

[8] M. Bozdal, M. Samie, S. Aslam, and I. Jennions, "Evaluation of can bus security challenges," *Sensors*, vol. 20, pp. 16–17, 04 2020.

[9] J. Cui, L. S. Liew, G. Sabaliauskaite, and F. Zhou, "A review on safety failures, security attacks, and available countermeasures for autonomous vehicles," *Ad Hoc Networks*, vol. 90, p. 101823, 2019, recent advances on security and privacy in Intelligent Transportation Systems.

[10] 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. New York, NY, USA: Association for Computing Machinery, 2017, p. 225–237.

[11] M. F. Schilling, "The surprising predictability of long runs," *Mathematics Magazine*, vol. 4 (20), pp. 141–149, 1986.

[12] F. Makri and Z. Psillakis, "On success runs of a fixed length in bernoulli sequences: Exact and asymptotic results," *in Computers and Mathematics with Applications*, vol. 61 (4), pp. 761–772, 2011.

[13] A. Philippou and F. Makri, "Successes, runs and longest runs," *in Statistics and Probability Letters*, vol. 4 (20), pp. 101–105, 1986.