Microcontroller Protocol for Secure Broadcast in Controller Area Networks

B.Vijayalakshmi¹, K. Kumar²

¹Final Year, Dept. of MCA, Vel Tech University, Avadi, Chennai – 600 062, Tamil Nadu, India.
²Assistant Professor, MCA Department, Vel Tech University, Avadi, Chennai – 600 062, Tamil Nadu, India.

ABSTRACT

Controller Area Network is a bus commonly used by controllers inside vehicles and in various industrial control applications. In the past controllers were assumed to operate in secure perimeters, but today these environments are well connected to the outside world and recent incidents showed them extremely vulnerable to cyber-attacks. To withstand such threats, one can implement security in the application layer of CAN. Here we design, refine and implement a broadcast authentication protocol based on the well known paradigm of using key-chains and time synchronization, a commonly used mechanism in wireless sensor networks, which allows us to take advantage from the use of symmetric primitives without the need of secret shared keys during broadcast. But, as process control is a time critical operation we make several refinements in order to improve on the authentication delay. For this we study several trade-offs to alleviate shortcomings on computational speed, memory and bandwidth up to the point of using reduced versions of hash functions that can assure ad hoc security. To prove the efficiency of the protocol the potential exists for an automotive ECU to be infiltrated by an attacker, who can then potentially gain access, via a serial communication bus, to an array of other ECUs. guarantee data integrity) and send their messages in the clear. Hence, room for fraudulent communications between ECUs exists. For example, in the CAN protocol, masquerade attacks followed by replay attacks

ECU pretending to be another ECU by sending/replaying a message the ECU is not entitled to send) are likely to happen as messages exchanged in a CAN network are broadcast from one ECU to the rest of the ECUs in the network. In fact, the receiver cannot verify the identity of the sender of the message as an attacker could have pretended to be someone else (and therefore sending a message with an ID the pretender was not configured to send in the first place). Again, this scenario is called a masquerade-based attack which then leads to a possible “replay” attack as the attacker, by pretending to

The state of the art processes, methods, and tools We are convinced that security can be taken into account in the early phases of the development cycle of automotive electronics systems, both by enforcing software programming standards that prevent soft-ware defects that may enable cyber-attacks, as well as by implementing security mechanisms such as authentication that enable the validation of the identity of the sender to avoid potentially harmful messages to be replayed/transmitted across the communication network. However, even for known vulnerabilities, one has to perform a cost versus benefits analysis as
the communication data rates available are very limited—it is necessary to evaluate whether a full authentication-based solution that addresses security concerns is compatible with performance and resource cost constraints that are typical of automotive embedded systems and specifically of the predominant communication protocols used in the vehicle (e.g., CAN has very limited data rates between 33kbps and 500kbps). In fact, authentication mechanisms typically require large amounts of processing power, memory, and bandwidth, in addition to those already reserved for the messages that are exchanged across ECUs. As more bytes need to be transmitted, current bus technologies may not be sufficient given their already limited available bandwidth.

Authentication mechanisms have been proposed in the literature. The TESLA protocol [3–5] uses a time-delayed release of keys for authentication. A receiver can check the Message Authentication Code (MAC) after receiving the key used to compute the MAC. To guarantee security, the protocol needs to maintain global time and make sure that a receiver gets a message before the corresponding key is re-released. In [6–8], the authors emphasize the con-straints in an embedded network and consider a time-triggered (i.e., global time is available) broadcast pro-to-col. Even with the features proposed for reducing the number of bits transmitted and for achieving fault tolerance, two major challenges exist in applying these approaches to the CAN protocol. First, the bandwidth available in the CAN protocol is very limited. Second, there is no notion of global time in the protocol. The challenge for OEMs in the automotive industry is to design a security mechanism for CAN with high security, combined with minimal communication overhead, high fault tolerance, low cost, and no global synchronization clock.

In this paper, we describe a security mechanism that addresses the requirements stated earlier. Specifically, our mechanism can be used to retrofit the CAN protocol to protect it from cyber-attacks such as masquerade and replay attack with as low as possible overhead, and high degree of tolerance to faults.

We address the low cost requirement by providing a software-only solution with no additional hardware required. We focus on the CAN protocol because it is the most used serial data protocol in current in-vehicle networked architectures, and it will likely be used for a long time. We define the attack scenarios that our security mechanism addresses, namely masquerade and replay. We focus on a security mechanism based upon message authentication and symmetric secret keys. Our mechanism leverages and modifies the work described in [6–8] as we introduce the concept of counters to implement time-stamping of the message signatures (MACs) in order to over-come the lack of global time in the CAN protocol. We do not focus on the initial security critical key assign-ment and distribution as this aspect, although very important, is already being mentioned in [6]. Instead, we focus on run-time authentication both in the system steady state (after ignition key-on and the security secret keys have been distributed to the ECUs) and during running resets experienced by some of the ECUs in the system (when counters are potentially out of synchronization). Regarding resets, we dis-tinguish between ECU running resets or any other ECU expected low-power modes that occur at rates that do not allow storing in non-volatile memory (flash) the most recent sending and receiving counters (needed for authentication) as this would lead to the flash being non-operational (e.g., due to burn-ing). We introduce two mechanisms that cope with these scenarios, which involve either an ECU that heals itself or a more drastic system-wide counter re-set (or re-synchronization). We provide an analysis of the trade-offs and the benefits versus drawbacks of both approaches. We also consider potential network faults that could hinder the effectiveness of our security mechanism—we provide a security mechanism that is fault tolerant. Finally, as we are constrained by data rates and by costs, we have defined a software-only mechanism that does not require additional hardware. As security has a cost in terms of performance (because of the additional bits needed for signatures and counters) and in terms of potential hazards that may occur due to poor performance, we also work on exploring trade-offs between degree of security and other metrics such as resource utilization. Experimental results show that our security mechanism can achieve high security level without introducing high communication overhead in terms of bus load and message latency.

The paper is organized as follows: Section II defines the system and attacker model; Section III presents the existing mechanisms, their limitations, our pro-posed security mechanism, and an evaluation of the impacts of the security mechanism on the system bus load and the message latency; Section IV shows the experimental results, and Section V concludes this paper.

II. SYSTEM AND ATTACKER MODEL

We adapt the terminology from [9] to the automotive use case, where a node is one of the computers (ECUs) connected to the other ECUs in the vehicle via a serial data communication bus to provide the following definitions of attack scenarios:

- **Modification**: an unauthorized node changes...
existing data (e.g., a sender node modifies the data portion of a communication frame to be transmitted).

- **Fabrication**: an unauthorized node generates additional data (e.g., a sender node creates a new frame with an ID that the node is not authorized to transmit).
- **Interception**: an unauthorized node reads data (e.g., a receiver node accepts a message with an ID that is not supposed to accept and reads the data portion of the frame).
- **Interuption**: data becomes unavailable (e.g., a sender node sends high priority frames over the communication bus at a very high rate making it impossible for other frames to be transmitted).

For the sake of our discussion, we generalize modification and fabrication as an unauthorized write of data by a node, an interception attack as an unauthorized read by a node, and an interruption attack as a Denial-of-Service (DoS) attack. We now define the following properties:

- **Data integrity**: data is not changed (written) or generated by an unauthorized node.
- **Confidentiality**: data is not read by an unauthorized node.
- **Authentication**: a receiver or a sender is who it claims to be.
- **Non-repudiation**: a sender ensures that a receiver has received the message, and a receiver is sure about the identity of a sender.

For automotive electronics systems and the CAN protocol, data integrity and authentication are very relevant properties which are suitable to our software-only security mechanism solution. To prevent an interruption attack, hardware protections are required as, because of the very same nature of the CAN protocol (broadcast and multi-master with arbitration), a malicious node can freely read and write data from/to the bus. Interruption attacks are outside of the scope of our work.

Before introducing our attacker model, we first state our assumptions, and provide definitions about our system model as follows:

**Assumption 1.** The network architecture has only one CAN bus, and all ECUs are connected to the bus itself.

**Definition 1.** A node is an ECU.

**Definition 2.** The sender of a message is the node sending the message.

**Assumption 2.** A sender sends a message by broadcasting it on the CAN bus.

**Definition 3.** A receiver of a message is a node receiving the message and accepting it by comparing the message ID to the list of its acceptable message ID's.

Note that CAN is a broadcast protocol, so every node “receives” the message, but only receivers (as we have defined them) accept the message.

**Assumption 3.** A node can use volatile (RAM) and/or non-volatile (FLASH) memory to store data. Data stored in RAM is no longer available after a node reset; data in FLASH is available after a node resets.

To describe our attacker model, we use a networked architecture topology as in Figure 1. Although in CAN, any node can play the role of sender and receiver in different bus transactions, for illustration purposes, we assume that node 1 is a sender node and node 3 is a receiver node. We also assume that nodes 1 and 2 are legitimate nodes. In Figure 1, if malicious software takes control of node 4, it can access any data stored in RAM and FLASH, including data used to implement a security mechanism (e.g., shared secret keys). It is also possible that an attacker uses a node (node 3) that has been added to the network (e.g., to perform diagnostics on the network this node could be laptop running diagnostic software and connected to the network using the CAN adapter interface); in this case, the malicious software also has access to the RAM and FLASH memory. However, no critical data (e.g., shared secret keys) is stored in RAM and FLASH in the first place.

**Definition 4.** A strong attacker is an existing node where malicious software is able to gain control with full access to any critical data.

**Definition 5.** A weak attacker is a node where malicious software is able to gain control but no critical data is available (mainly because it was never stored in memory).

**Definition 6.** A legitimate node is a node which is neither a strong attacker nor a weak attacker.
For example, in Figure 1, \( N_3 \) and \( N_4 \) are strong and weak attackers, respectively, and \( N_1 \) and \( N_2 \) are legitimate nodes. The possible attack scenarios that \( N_3 \) and \( N_4 \) can carry out and that we are addressing with our solution are:

<table>
<thead>
<tr>
<th>Types</th>
<th>Strong Attacker ( N_3 )</th>
<th>Weak Attacker ( N_4 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modification or Fabrication</td>
<td>Scenario 1</td>
<td>Scenario 2</td>
</tr>
<tr>
<td>Replay</td>
<td>Scenario 3</td>
<td>Scenario 4</td>
</tr>
</tbody>
</table>

In the table, we describe the scenario in which a message is supposed to be send by a legitimate sender (\( N_1 \)). However, \( N_3 \) and \( N_4 \) try to alter this situation with either a strong or weak attack. Again, we are not addressing attacks such as DoS as they would require additional hardware—our proposed solution is software-only. We now explain the scenarios as follows:

- Scenario 1: this is possible if important/secret data between \( N_1 \) and \( N_2 \) has been stored in RAM or FLASH of \( N_3 \). For example, if important/secret data is shared and used by every node in the network\(^3\), then \( N_1 \) can use the data stored in RAM or FLASH and pretend to be \( N_1 \) to send a new message to \( N_2 \) (fabrication).

- Scenario 2: this is no threat because no important/secret data is stored in RAM or FLASH of \( N_3 \).

- Scenario 3: this is possible if \( N_3 \) reads a message from the CAN bus and then writes the same message to the CAN bus without any modification. Note that, in this case, \( N_3 \) does not need to get important/secret data between \( N_1 \) and \( N_2 \), e.g., a secret pair-wise key as in Figure 2, because \( N_2 \) will just accept the message thinking it was sent by \( N_1 \).

- Scenario 4: same as Scenario 3.

We now define a masquerade and replay attack and show how we can prevent it as follows [7]:

Definition 7. In a masquerade attack, an attacker (strong or weak) sends a message in which it claims to be a node other than itself.

Note that a masquerade attack can lead to a fabrication attack, a modification attack, or as a special case, a replay attack:

Definition 8. A replay attack is enabled by a masquerade attack, and the node in order to be successful, needs rst to pretend to be another node. In the case of CAN, in a replay attack a node transmits a copy (replays) of a message it has received from the CAN bus. The message is not modi ed or altered. It is merely sent to other nodes by a node that is not entitled to send it. The other nodes have tables that match the message id to the sender and therefore, determine the identity of the sender but have no provision to au-thenticate it.

Since CAN is a broadcast protocol, both a strong and weak attacker can successfully carry out a masquerade/replay attack if no security mechanism is put in place, or even if pair-wise keys are used as the attacker would not need them to successfully carry on the attack. Before introducing some basic security mechanisms, we also provide a definition of a false acceptance and a false rejection as follows:

Definition 9. A false acceptance is the scenario that a node accepts messages which it should reject.

Definition 10. A false rejection is the scenario that a node rejects messages which it should accept.

By the definition, a successful attack implies a false acceptance.

### III. SECURITY MECHANISMS

In this section, we will first introduce some basic authentication mechanisms and describe the existing work in this area in more detail. Then, we will show the challenges in implementing a security mechanism for CAN and how we can overcome these difficulties with our proposal. Finally, we will provide our counter-based implementation, reset mechanisms, and some detailed analysis of their performance vs. security levels achieved. We now provide a few additional definitions that we will use in the rest of the paper.

<table>
<thead>
<tr>
<th>Notations</th>
<th>Explanations</th>
</tr>
</thead>
<tbody>
<tr>
<td>( i )</td>
<td>the ID of a node</td>
</tr>
<tr>
<td>( j )</td>
<td>the ID of a node</td>
</tr>
<tr>
<td>( k )</td>
<td>the ID of a message</td>
</tr>
<tr>
<td>( N_i )</td>
<td>the node with ID ( i )</td>
</tr>
<tr>
<td>( M_k )</td>
<td>the message with ID ( k )</td>
</tr>
<tr>
<td>( n )</td>
<td>the number of nodes</td>
</tr>
<tr>
<td>( s_k )</td>
<td>the number of receivers of ( M_k )</td>
</tr>
<tr>
<td>( f )</td>
<td>the function to compute a MAC</td>
</tr>
<tr>
<td>( T )</td>
<td>the time</td>
</tr>
<tr>
<td>( k_{i;j} )</td>
<td>the shared secret key of ( N_i ) and ( N_j )</td>
</tr>
<tr>
<td>( A_k )</td>
<td>the MAC computed by a receiver</td>
</tr>
<tr>
<td>( C_i )</td>
<td>the counter stored in ( N_i ) for ( M_k )</td>
</tr>
<tr>
<td>( C_{M_i} )</td>
<td>the most signiﬁcant bits (MSBs) of a counter</td>
</tr>
<tr>
<td>( c_L )</td>
<td>the least signiﬁcant bits (LSBs) of a counter</td>
</tr>
</tbody>
</table>
1 BASIC AUTHENTICATION

![Image 1](https://example.com/image1)

![Image 2](https://example.com/image2)

![Image 3](https://example.com/image3)

Figure 2. Pair-wise secret key distribution.

Basic authentication is based on sharing a secret key between a sender N1 and a receiver N2 and computing a Message Authentication Code (MAC) [6] which is essentially a signature of a message. A key K_{i,j} is the shared secret key stored in N1 and N2 and only known by N1 and N2. For the sake of the discussion and without loss of generality, we assume a pair-wise secret key assignment (an example is shown in Figure 2). N1 and N2 perform the following steps to send and receive a message M_i:

**Sender (N_1)**
1. Send M_i and A_{k,i}
2. \( k_i = f(M_i; K_{1,2}) \)

**Receiver (N_2)**
1. Receive M_i and A_{k,1}
2. \( A = f(M_i; K_{1,2}) \)
3. Accept M_i if and only if \( A = A_{k,1} \)

Note that the “1” of A_{k,1} means that N2 is the first and the only receiver of M_i. Even if N1 is a strong attacker, since the keys are assigned in a pair-wise fashion, N1 is not able to compute the MAC (as it is missing K_{1,2}) that is needed to attack N2 with a message that is supposed to be sent by N1. However, since in a broadcast protocol the message transmitted is read by any node in the network, and M_i and A_{k,1} are sent in the clear, N1 could read this data and resend it verbatim (essentially replay the same message). N2 is going to accept it as the MAC is a match. A possible solution to this problem is to use the concept of global time that allows time-stamping messages. If global time is adopted then N2 can prevent the attack from N1. An authentication mechanism with global time-stamping as follows:

**Sender (N_1)**
1. Get time \( T \)
2. \( A_{k,i} = f(M_i; T; K_{1,2}) \)
3. Send M_i and A_{k,1}

**Receiver (N_2)**
1. Receive M_i and A_{k,1}
2. Get sending time \( T \)
3. \( A = f(M_i; T; K_{1,2}) \)
4. Accept M_i if and only if \( A = A_{k,1} \)

As in the scenario explained earlier, if N1 wants to send M_i to N2, as it cannot retrieve K_{1,2} because it does not have it, it cannot compute the correct MAC. In addition, in case of a replay attack, if N1 replays the message it will do so using a MAC computed using an earlier time stamp that what N2 would use to compute the MAC. Therefore, the MACs cannot match, and N2 rejects the message. As we will show later in this paper, global time is not available in CAN and therefore we introduce monotonic counters to address replay attacks.

2 EXISTING WORK

The basic authentication mechanisms have been summarized in the above section, but there are still other alternatives and variations for authentication. A lot of existing work focuses on digital signatures. However, digital signatures have very high communication overhead, making them inapplicable or at least very difficult to use for CAN.

In [6–8], the authors emphasize the constraints in an embedded network and consider a time-triggered (i.e., global time is available) broadcast protocol. Since every node is a receiver, a transmitted message includes MACs for all receivers. Therefore, N1 and N2 perform the following steps to send and receive a message M_i:

**Sender (N_1)**
1. Get time \( T \)
2. \( A_{k,i} = f(M_i; T; K_{1,2}) \)
3. Send M_i and A_{k,1}

**Receiver (N_2)**
1. Receive M_i and A_{k,1}
2. Get sending time \( T \)
3. \( A = f(M_i; T; K_{1,2}) \)
4. Accept M_i if and only if \( A = A_{k,1} \)

The authentication operation using the for-loop uses \( n \) since the authors are using a comprehensive definition of receiver. This means that there are as many receivers as nodes in the network. Each receiver authenticates the message by first identifying the correct MAC that the receiver needs to compare to, based upon the information that maps each re-transmitted message to the unique sender of the message itself. Besides the authentication aspect, the authors have introduced other interesting features to their authentication mechanism to cope with the potentially limited communication bus data rate and provide fault tolerance. First, only a subset of the MAC bits are sent and used for authentication purposes, i.e., \( A \) and \( A_{k,1} \) in the above operations are replaced by \([A]_0\) and \([A_{k,1}]_0\) where \([i]\) is the truncation operation to \( i \) bits. The authors, in their analysis, assume that an unsafe state is reached only when at least \( k \) out of \( n \) most recently received messages are successfully attacked. Lastly, in their extension work
[8], the authentication is performed by different voting nodes.

3 CHALLENGES FOR CAN

Even with the features proposed for reducing the number of bits transmitted and achieving fault tolerance, two major challenges exist in applying the work just described to CAN. First, the bandwidth available in CAN is extremely limited. In fact, the maximum and nominal data rate of a CAN bus is only 500 kbps, while each 11-bit ID standard frame has a maximum total of 134 bits which include a maximum of 64-bit payload, 46 bits of overhead (e.g., including CRC bits), and 24 bits for bit-stuffing [10] in the worst case. If a security mechanism needs to add MACs to the original frame, as the original frame might have a 64-bit payload, the frame might have to be split in two or more frames. This may result in increasing bus utilization which may result in a degraded communication performance or even in a unschedulable system. Finally, as stated earlier, there is no global time in CAN (the global time is required in [3–8]).

4 OUR SECURITY MECHANISM

The key elements of our proposed security mechanism are stored in each node (in the volatile and non-volatile memory). The elements are: the ID table, the pair-wise symmetric secret keys, and message counters (receiving and sending). In the following, we use our definition of receivers (see Definition 3).

• ID table: unlike the approach described in [6–8], our mechanism does not use MACs for all nodes. On the contrary, a sender only computes as many MACs as the corresponding receivers of the transmitted message. This is done by maintaining a ID table in each node where each entry is indexed by a message ID — each entry contains the node ID of the sender and the list of the node ID's of the receivers. We define the ID table with the following function:

\[
\text{id}(i; n_k; r_{k,1}; r_{k,2}; \ldots ; r_{k,n_k}) = \text{ID-Table}(k);
\]

where \(k\) is the ID of \(M_k\), \(i\) is the ID of the sender of \(M_k\), \(n_k\) is the number of receivers of \(M_k\), and \(r_{k,s}\) is the ID of the \(s\)-th receiver of \(M_k\). A sender can check its ID table to determine how many MACs it must compute, what keys it should use, and what ordering of MACs it should attach with the message. A receiver can check the ID table to determine what key it should use and which MAC included in the received frame it should select. Again, the ad-antage of relying on ID tables is that our mechanism reduces the number of MACs because it considers only the receivers that are accept-ing the frame after CAN filtering, rather than considering the whole set of receivers that the frame is broadcast to. This can reduce the communication overhead considerably.

• Pair-wise secret key: a pair-wise key \(K_{ij}\) is “shared secret” between \(N_i\) and \(N_j\) for authentication. Every pair of nodes has a shared secret key which is not known by any other node. Therefore, any other node cannot modify or fabricate a message, but a replay attack is possible as explained earlier. Note that using pair-wise keys is only a basic key distribution method. If we want to further reduce the communication overhead, we could assign nodes to several groups where each node in a group shares a secret key. Of course, there is a trade-off between security and performance (minimizing communication overhead) in that the security level is diminished but the communication performance is improved.

• Message-based counter: a counter is used to replace the global time and prevent a replay attack. Each node maintains a set of counters, and each counter corresponds to a message, i.e., \(C_{i,k}\) is the counter stored in \(N_i\) for \(M_k\). If the node is the sender of \(M_k\), its counter value records the number of times that \(M_k\) is sent; if the node is the receiver of \(M_k\), its counter value records the number of times \(M_k\) has been received (and accepted after being authenticated). Therefore, if a malicious node replays a message, a receiver can check the corresponding receiving counter to see if a message is fresh or not. Because of a network fault, a receiving counter may not have the same value as that of its sending counter. In other words, it is possible that a node sends a frame, updates its sending counter, then a network fault occurs, e.g., the electrical bus has a transient fault, and thus the frame never reaches its destination. Therefore, the receiving node does not receive the frame and thus does not increase its receiving counter. This means that two counters are out of synchronization. However, our mechanism can deal with this scenario without any loss of security. We will explain this aspect later in the paper. We now provide the following additional definitions:

Definition 11. A sending counter for a message is the counter stored in its sender.

Definition 12. A receiving counter for a message is the counter stored in one of its receiver.

In our security mechanism, every node maintains its ID table, pair-wise keys, and counters. \(N_i\) and \(N_j\) perform the following steps to send and
receive a message \( M_i \):

**Sender (N)\)**

1. \( (i; n_i; r_{k;1}; r_{k;2}; \ldots ; r_{k,nk} ) = \text{ID-Table}(k) \)
2. \( r_{k;1} = C_{i,k} + 1 \)
3. \( n_{s} = C_{i,k} + \sum_{s=1}^{k} s \cdot C_{i,k} = f^{M,k} C_{i,k} K_{i;r_{k;1}} \)
4. Send \( M_i : A_{i;k;1}; A_{i;k;2}; \ldots ; A_{i;k,nk} \)

**Receiver (N)\)**

1. Receive \( M_i : A_{i;k;1}; A_{i;k;2}; \ldots ; A_{i;k,nk} \)
2. \( (i; n_i; r_{k;1}; r_{k;2}; \ldots ; r_{k,nk} ) = \text{ID-Table}(k) \)
3. Continue if only if find \( s; 1 \leq s \leq n_c; j = 1 \)
4. \( A = f(M_i; C_{i,k}; K_{i,j}) \)
5. Accept \( M_i \) and \( C_{i,k} = C_{j,k} \) if and only if \( A = A_{k;1} \)

Based on this mechanism, our security mechanism can protect any masquerade attack and replay attack. We prove our claim using the following three scenarios:

- If an attacker sends a message which is not supposed to be received by the receiver, then the receiver will reject the message in Line 6 by checking its ID table.
- If an attacker sends a message which is not supposed to be sent by the attacker, and it is a replay attack, then the receiver will reject the message in Line 2 by checking the counters.
- If an attacker sends a message which is not supposed to be sent by the attacker, and it is not a replay attack, then the receiver will reject the message in Line 12 by comparing the MACs.

5 COUNTER IMPLEMENTATION

These operations can meet the requirements stated by our problem formulation. However, the number of bits used for the counter must be explored. If the number of bits is not sufficient during the lifetime of a vehicle, then the counter may overflows. For example, if the counter stored at the receiving side overflows and resets to zero, then the replay attack may succeed as the attacker just needs to wait for this event to happen, and therefore resend a counter which is larger than the reset counter stored in the receiver; if the number of bits used for the counter is too large, then the bus will be overloaded. Therefore, we propose a solution where the counter \( C \) is divided into two parts: the most significant bits (MSBs) \( C^M \) and the least significant bits (LSBs) \( C^L \)—only \( C^L \) is transmitted with the message. The steps performed by \( N_i \) and \( N_j \) are similar, but only \( C_{i,k}^L \) is transmitted:

**Authenticated-Sending(M)\)**

1. \( (i; n_i; r_{k;1}; r_{k;2}; \ldots ; r_{k,nk} ) = \text{ID-Table}(k) \)
2. \( A_{k;1} = C_{i,k} + 1 \)
3. \( A_{k;1} \) for \( s = 1 \)
4. \( A_{k;1} = n_{s} + n_{s} + 1 \)
5. Send \( M_i : A_{i;k;1}; A_{i;k;2}; \ldots ; A_{i;k,nk} \)

**Authenticated-Receiving(M)\)**

1. \( (i; n_i; r_{k;1}; r_{k;2}; \ldots ; r_{k,nk} ) = \text{ID-Table}(k) \)
2. Find \( s \) such that \( 1 \leq s \leq n_c \) and \( j = j_{max} \)
3. If \( s \) is not found, return Reject;
4. If \( C_{k;1} > C_{i,k} \)
5. \( A = f(M_i; C_{i,k}; K_{i,j}) \)
6. Return Accept;
7. Return Reject;
8. If \( A = A_{k;1} \)
9. Return Accept;
10. Return Reject;
11. If \( C_{k;1} = C_{i,k} - 1 \)
12. Return Accept;
13. Return Reject;
14. If \( A = f(M_i; C_{i,k}; K_{i,j}) \)
15. Return Accept;
16. Return Reject;
17. Return Accept;
18. Return Reject;
19. Return Reject;

For more details, the reader should see Figure 3. If \( C_{i,k}^L \) \( \neq C_{j,k}^L \), then this is the same scenario as that of the original mechanism; if \( C_{i,k}^L \leq C_{j,k}^L \), then the receiver will use \( C_{i,k}^L + 1 \) to compute the MAC. If there is a replay attack, then the receiver will test \( C_{j,k}^L = C_{i,k}^L \) to be true and use \( C_{j,k}^L + 1 \) to compute the MAC which will be different from the one transmitted in the replayed message. The receiver will fail the test comparing the stored computed MAC and the received MAC and will reject the message.

The advantage of using this mechanism is that we can reduce the communication overhead without any loss of security. Of course, if the receiver consecutively misses several messages due to a network fault, it may reject a message although there is no attack in place, as its receiving counter may not be up-to-date (out of synchronization). However, the probability that a counter is out of synchronization is very low. If a counter is divided into \( C^M \) and \( C^L \) and the probability of a network fault is \( q \), the probability that a counter is out of synchronization is \( q^{2^{C^L}} \). For example, if \( |C^L| = 3 \) and \( q = 0.1 \), the probability that a counter is out of synchronization is only 0.1\(^3\). Even if this scenario occurs and the computed MAC would not match although it would pass the counter test, the receiver will continue rejecting messages (false rejection). Although this scenario is not optimal, a counter out of synchronization is a better option than a successful attack. In addition, we ad-dress this...
potential issue by providing counter reset mechanisms. This is the focus of the next section in the paper.

6 COUNTER RESET MECHANISMS

A counter reset mechanism is required to deal with an ECU hardware reset or with counters that are out of synchronization because of a network fault. There are two types of hardware resets. First, either an ECU may reset as expected, e.g., as it goes into a low power mode as a result of a specific driving mode in which some ECU's are shut off to reduce the energy usage, or the ECU experiences an unexpected hardware reset due to a power failure. Regardless of the reason why an ECU resets, the rate at which the resets occur or the minimum time interval between them might be too short to allowing storing critical data into FLASH which could be restored at a later time, as storing data in the FLASH too frequently (at a rate that is higher than of the expected maximum rate of resets) may lead to burning the FLASH itself. Therefore, we have devised mechanisms that deal with scenarios where critical data such as updated counter values cannot be stored in FLASH at a rate that makes them sufficiently up-to-date (or close to) to avoid excessive false rejections on the receiver side when they are later restored into RAM. When data can be copied to FLASH the mechanism is simple. Before an expected shutdown, or change of power state, the ECU copies and stores the relevant data in FLASH from RAM. At wake-up, the ECU restores the data from FLASH into RAM. However, unexpected shutdowns can occur when a hardware failure occurs, or there is a lack of power, etc. In this case, it is not safe to assume that critical data was stored in FLASH and that can be restored. Therefore, provisions have to be put in place to bring back the ECU, and therefore the system, to a secure state (e.g., with counter values that prevent attacks). Our mechanisms that deal with unexpected hardware resets include "node self-healing" and "network-wide" counter resets. The mechanisms provide trade-offs between security levels and communication overhead. In the following, we describe the self-healing mechanism operations performed by a node that has experienced a hardware reset.

1. The node sets a FLAG variable to zero.
2. The node stores its counters into FLASH every P seconds. The time interval P is a function of the FLASH technology.
3. If a node is experiencing an expected hardware reset, then the node tries to store the latest counters value from RAM to FLASH before shutting down. If the operation is successful (it may not be if the FLASH controller refuses to allow it because of potential burning), then FLAG should be set to 1. If not, the remain-ing steps are the same of those taken in case of an unexpected hardware reset due to a power failure.
4. If a node reset unexpectedly, nothing can be guaranteed including storing data to FLASH, therefore the FLAG stays at zero.
5. When a node wakes up, if FLAG = 1, it restores all counters from FLASH and set FLAG = 0; if FLAG = 0, it restores all counters from FLASH (last counters saved) and increase them by Q, and stores them into FLASH.

\[ P \] is a parameter that depends upon the FLASH technology. There is a trade-off between data freshness and expected life of the FLASH memory. \( Q \) is the upper bound of the number of messages that could be sent within the time interval P to prevent a replay attack—different counters can be associated with different values of \( Q \) for different messages.

Since the value of \( Q \) is an estimate provided by the designer of the number of messages instances received during \( P \), it is possible that this value is not the real upper bound or worst case number of message in-stances sent during \( P \). Hence, a larger \( Q \) value than the real one may lead to false rejections, meaning to a situation where a receiving node has a receiving counter that is higher than the counter being received although it should not be. In this case, the receiving node may reject message instances even if it should not until the sender counter reaches the receiver stored counter value. Conversely, if \( Q \) is smaller than what it should be, then the receiver will accept message instances it should not (false acceptances). In both cases, the designer is expected to tune the value of \( Q \) off-line.

The advantage of this mechanism is that at wake-up following a node re-set, a node resets its counters by itself without the need of additional messages to reset the counters of other nodes. Therefore, the
communication overhead is minimized as no network-wide counter synchroniza-
tion is necessary. However, as the parameter \( Q \) is an estimate, potential
false rejections or, even worse, false acceptances may occur.

Besides the self-healing reset mechanism, we also pro-
pose a network-wide reset mechanism. The key con-
cepts are:

- A RESET message to set all counters of all
  nodes to 0.
- A REQUEST message to achieve fault toler-
  ance.
- New session keys to prevent replay attacks.

In this mechanism, because every counter is reset to
0, new session keys are required; otherwise, an at-
tacker could successfully replay-attack. Therefore, a
random generated number needs to be included in a
RESET message, as it is used to generate the new
session key for each node. We can further divide this
into two possible approaches. The first one is a “dy-
namic” network reset where any node experiencing a
reset can generate a random number and send a
RESET message to all other nodes. The second ap-
proach is a “static” network reset where only one spe-
cial master node can generate a random number and
send a RESET message to all other nodes.

**Figure 4.** The finite state machine of a node in the
dynamic network reset.

The finite state machine of a node in the dynamic
network reset is shown in Figure 4. This approach
has the following features:

- Every node needs to maintain a variable FLAG
to indicate if it is the last node generating the
random number \( X \) and sending the RESET
message.

- If a node experiences a reset (goes to the re-set state),
then it will set all counters to 0, set
FLAG to 1, generate a random number \( X \) and
its new session keys, and send a RESET mes-
 sage with \( X \).

- If a node receives a RESET message, then it
will set all counters to 0, set FLAG to 1, and
generate its session keys.

- If a node finds itself out of synchronization
(missing a RESET message due to network
fault), then it will send a REQ message to ask
for going back to synchronization.

If a node receives a REQ message, then it will
check if FLAG is 1. If yes, it is the last node
generating \( X \) and sending the RESET message, so it
will resend a RESET message.

**Figure 5.** The finite state machine of a master node
in the static network reset.

**Figure 6.** The finite state machine of a non-master
node in the static network reset.

The finite state machine of a master node in
the static network reset is shown in Figure 5; the
finite state machine of a non-master node in the static
network reset is shown in Figure 6. The di-

The differences between static and dynamic resets are as follows:

- A node does not need to maintain a variable
FLAG because only the master node can gener-
ate a random number and send a RESET mes-
 sage.

- A REQ0 message is used by a non-master node
to ask the master node to reset the network.

- If a non-master node experiences a reset, then
it will send a REQ0 message and wait for a
 RE-SET message.

- If a master node receives a REQ0 message, it
will set all counters to 0, generate a random
number \( X \) and its session keys, and send a RE-
SET message with \( X \).
Although the network-wide reset mechanism can guarantee that there is no false rejection or successful replay attack, it may determine high transient bus peak loads due to the increasing traffic created by the messages used to reset the counters in every node.

To this point, we have proposed a self-secure criticality level of each message.

7 ANALYSIS

We show how the security mechanism has an impact on the system bus load and message latencies by formulating the problem as a feasibility analysis problem. The system model includes the following parameters:

- \( n \): the number of messages.
- \( q \): the probability that a message is missing due to a network fault.
- \( R \): the bus data rate.

The following message \( M_i \) parameters are defined:

- \( n_i \): the number of the message receivers.
- \( R_i \): the message rate (frequency, as the inverse of its period).
- \( S_i \): the message original size.
- \( L_i \): the upper-bound of the total length of MACs and LSB of the counter.
- \( C_i \): the lower-bound of the length of LSB of the counter.
- \( P_i \): the upper-bound of the probability of a successful attack.
- \( Q_i \): the upper-bound of the probability that a counter is out of synchronization.

If \( M_i \) is not a security-critical message, then \( C_i = 0 \) and \( P_i = Q_i = 1 \).

We define the following decision variables:

- \( x_i \): the length of the MAC for \( M_i \).
- \( y_i \): the length of the LSB of the counter for \( M_i \).

We define several constraints for \( M_i \) as follows:

- The length of LSB of the counter should be larger than or equal to \( C_i \).
- The probability of a successful attack should be smaller than or equal to \( P_i \).
- The probability that a node is out of synchronization should be smaller than or equal to \( Q_i \).

The constraints in mathematical forms are defined as follows:

\[
x_k + y_k \leq L_k; \quad y_k \geq Q_k; \quad x_k \leq P_k; \quad q^2 k \leq Q_k;
\]

The last two constraints also define the probability of a false acceptance (a node accepts messages which it should reject) and a false rejection (a node rejects messages which it should accept). We can easily derive the minimal values of \( x_k \) and \( y_k \) and then compute the message latency using the equation [11]:

\[
b = b + \sum_{i \in R(k)} \frac{S_i + x_i + y_i}{2hp(k)} R
\]

where \( k \) is the latency of \( hP(k) \), \( b = \max \{ \text{message} \} \). 

and \( hp(k) \) is the index set of messages with higher priorities than \( M_k \). By using a traditional fix-point calculation, the latency is computed through an iterative method until convergence (if a solution exists).

IV. EXPERIMENTAL RESULTS

In this section, we show how our security mechanism impacts on the system bus load and message latency. Since there is no global time in CAN, the approaches in [3-8] are not applicable to CAN networks. We used a test case with 17 security-critical messages among 138 messages, and \( q = 0:1 \), \( R = 500 \) (kbps), \( L_k = 32 \) (bits), \( C_k = 1 \) (bit) for all security-critical messages. Table 1 and Table 2 show the relative bus load and average latency with different values of \( P \) and \( Q \), where \( P_k = P \) and \( Q_k = Q \) for all \( k \), under the assumptions that the \( n_i \) is 1 or 3. The number of receivers was not known at the time of our experiments, so we have used a simple assumption. If this information is provided, more general experiments can be done by assigning different values for \( P_k \) and \( Q_k \) for different \( k \). Again, the main purpose of this paper is to provide a security mechanism and show how the
We described a security mechanism that can be used to retro-fit the CAN protocol to protect it from cyber-attacks such as masquerade and replay attacks. The mechanism is suitable for this protocol because it has a low communication overhead and does not need to maintain global time. Besides, the solution is software-only, hence, it is not overly expensive to implement. Experimental results showed that our security mechanism can achieve high security level without introducing high communication overhead in terms of bus load and message latency.

V. CONCLUSIONS

VI. ACKNOWLEDGEMENTS

We would like to thank our Guide Mr K. Kumar assistant Professor Vellore Institute of Technology for the valuable discussions and feedback.
References


