Title
Cyber-security for the Controller Area Network (CAN) communication protocol

ISBN
9780769550145

Authors
Lin, CW
Sangiovanni-Vincentelli, A

Publication Date
2012

DOI
10.1109/CyberSecurity.2012.7

Peer reviewed

Cyber-Security for the Controller Area Network (CAN) Communication Protocol

Chung-Wei Lin
University of California, Berkeley
Email: cwlin@eecs.berkeley.edu

Alberto Sangiovanni-Vincentelli
University of California, Berkeley
Email: alberto@eecs.berkeley.edu

***Abstract*—We propose a security mechanism to help prevent cyber-attacks (masquerade and replay) in vehicles with architecture based on Controller Area Network (CAN). We focus on CAN as it will likely continue being used in upcoming in-vehicle architectures. The CAN protocol contains no direct support for secure communications. Retrofitting the protocol with security mechanisms poses several challenges given the very limited data rates available (*e.g.*, 500kbps) since bus utilization may significantly increase. In this paper, we focus on a security mechanism which keeps the bus utilization as low as possible. Through our experimental results, we show that our security mechanism can achieve high security levels while keeping communication overheads (*e.g.*, bus load and message latency) at reasonable levels.**

I. Introduction

Modern automotive electronics systems are distributed as they are implemented with software running over networked Electronic Control Units (ECU) communicating via serial buses and gateways. Most systems (but not all; indeed, the automotive industry has started to take actions to prevent tampering with calibration parameters in engine control applications) have not been designed with security in mind. In addition, in the majority of the cases, there was little or no interest for hackers to compromise them. The only exception known so far is the after-market community that tampers with engine calibrations to increase engine's performance.

Recently, Koscher *et al.* demonstrated in [2] that 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. However, no security aspect is included as part of the hardware and software architecture development process and no standard communication protocol has any built-in provisions to prevent or mitigate attacks.

Communication networks are vulnerable as they enable unauthorized access in a relatively straightforward manner as all the communications between the ECUs in the vehicle are performed with no authentication [11]. Authentication mechanisms ensure that sender and receiver identities are not compromised and thus, the sender and the receiver are who they are claiming to be. Unfortunately, current communication network protocols, including Controller Area Network (CAN), FlexRay, MOST, and LIN have no authentication (or at best have CRC mechanisms to 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 (an 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 be someone else, is replaying verbatim the same message it has received although not entitled to do so.

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 software 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 [4], [5], [6] 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 released. In [8], [9], [10], the authors emphasize the constraints in an

embedded network and consider a time-triggered (*i.e.*, global time is available) broadcast protocol. 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 retro-fit 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 do not focus on the initial security critical key assignment and distribution as this aspect, although very important, is already being mentioned in [8]. 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). 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 proposed security mechanism; Section IV shows the experimental results, and Section V concludes this paper.

II. System and Attacker Model

We adapt the terminology from [7] 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).
- *Interruption*: 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[1].

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 N_1 is a sender node and N_2 is a receiver node. We also assume that N_1 and N_2 are legitimate nodes. In Figure 1, if malicious

[1]Our definition of a receiver is a specialization of the definition of a receiver in CAN where instead a receiver can also reject a message.

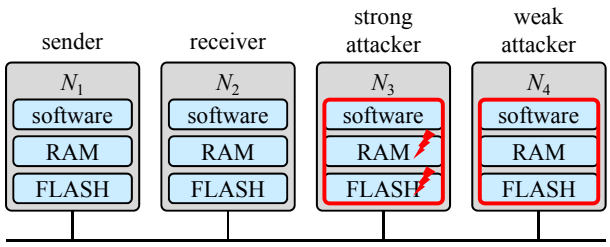

Fig. 1. Attacker Model.

Types	Strong Attacker N_3	Weak Attacker N_4
Modification or Fabrication	Scenario 1	Scenario 2
Replay	Scenario 3	Scenario 4

TABLE I
Attack scenarios that an attacker pretends as a legitimate sender (N_1) and sends a message to a legitimate receiver (N_2).

software takes control of N_3, 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 (N_4) 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. We are now ready to provide some definitions as follows:

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 shown in Table I. 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[2], then N_3 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: there is no threat because no important/secret data is stored in RAM or FLASH of N_4.
- 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 [9]:

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 first 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 modified 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 authenticate 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

We provide a few additional definitions that we will use in the rest of the paper (see Table II).

A. Existing Work

A lot of existing work focus on digital signatures. However, digital signatures have very high communication overhead, making them inapplicable or at least very difficult to use for CAN.

[2]For example, if the nodes in the network share the same secret key. This is a different scenario from the scenario in Figure 2 where nodes share secret keys in a pair-wise fashion.

Notations	Explanations
i	the ID of a node
j	the ID of a node
k	the ID of a message
N_i	the node with ID i
M_k	the message with ID k
n	the number of nodes
n_k	the number of receivers of M_k
$r_{k,s}$	the ID of the s-th receiver of M_k
f	the function to compute a MAC
T	the time
$K_{i,j}$	the shared secret key of N_i and N_j
$A_{k,s}$	the MAC for the s-th receiver of M_k
A	the MAC computed by a receiver
$C_{i,k}$	the counter stored in N_i for M_k
C^M	the most significant bits (MSBs) of a counter
C^L	the least significant bits (LSBs) of a counter

TABLE II
Table of notations.

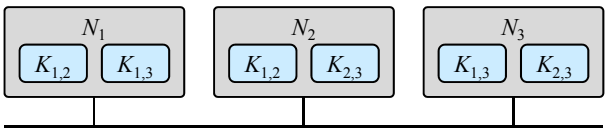

Fig. 2. Pair-wise secret key distribution.

In [8], [9], [10], a pair-wise secret key assignment (an example is shown in Figure 2) is used. 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[3], a transmitted message includes MACs for all receivers. Therefore, N_1 and N_2 perform the following steps to send and receive a message M_k:

Sender (N_i)	
1	Get time T
2	$\forall j, 1 \leq j \leq n, A_{k,j} = f(M_k, T, K_{i,j})$
3	Send $M_k, A_{k,1}, A_{k,2}, \ldots, A_{k,n}$

Receiver (N_j)	
1	Receive $M_k, A_{k,1}, A_{k,2}, \ldots, A_{k,n}$
2	Get sending time T
3	Get i where N_i is the sender of M_k
4	$A = f(M_k, T, K_{i,j})$
5	Accept M_k if and only if $A = A_{k,j}$

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 received message to the unique sender of the message itself. Besides the authentication aspect, the authors have also 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 $A_{k,j}$ in the above operations are replaced by $[A]_l$ and $[A_{k,j}]_l$ where $[]_l$ is the truncation operation to l 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 [10], the authentication is performed by different voting nodes.

[3]The authors use the more comprehensive version of a receiver where a receiver can accept or reject a message.

B. 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 500kbps, 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 [1] 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 [4], [5], [6], [8], [9], [10]).

C. 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 [8], [9], [10], our mechanism does not use MACs for all nodes. On the contrary, a sender only computes as many MACs as the corresponding receivers[4] 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:

$$(i, n_k, r_{k,1}, r_{k,2}, \ldots, r_{k,n_k}) = \text{ID-Table}(k),$$

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 advantage of relying on ID tables is that our mechanism reduces the number of MACs because it

[4]In our "specialized" meaning.

considers only the receivers that are accepting 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 $K_{i,j}$ 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 a 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_k:

Sender (N_i)	
1	$(i, n_k, r_{k,1}, r_{k,2}, \ldots, r_{k,n_k}) =$ ID-Table(k)
2	$C_{i,k} = C_{i,k} + 1$
3	$\forall s, 1 \leq s \leq n_k, A_{k,s} = f(M_k, C_{i,k}, K_{i,r_{k,s}})$
4	Send $M_k, C_{i,k}, A_{k,1}, A_{k,2}, \ldots, A_{k,n_k}$

Receiver (N_j)	
1	Receive $M_k, C_{i,k}, A_{k,1}, A_{k,2}, \ldots, A_{k,n_k}$
2	$(i, n_k, r_{k,1}, r_{k,2}, \ldots, r_{k,n_k}) =$ ID-Table(k)
3	Continue if and only if find $s, 1 \leq s \leq n_k, j = r_{k,s}$
4	Continue if and only if $C_{i,k} > C_{j,k}$
5	$A = f(M_k, C_{i,k}, K_{i,j})$
6	Accept M_k and $C_{j,k} = C_{i,k}$ if and only if $A = A_{k,s}$

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.

D. 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 overflow. 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 $C^L_{i,k}$ is transmitted:

Authenticated-Sending(M_k)	
1	$(i, n_k, r_{k,1}, r_{k,2}, \ldots, r_{k,n_k}) =$ ID-Table(k);
2	$C_{i,k} = C_{i,k} + 1$;
3	**for** $s = 1$ **to** n^r_k
4	$\quad A_{k,s} = f(M_k, C_{i,k}, K_{i,r_{k,s}})$;
5	Send $M_k, C^L_{i,k}, A_{k,1}, A_{k,2}, \ldots, A_{k,n_k}$;

The Authenticated-Receiving is shown in Figure 3. If $C^L_{i,k} > C^L_{j,k}$, then this is the same scenario as that of the original mechanism; if $C^L_{i,k} \leq C^L_{j,k}$, then the receiver will use $C^M_{j,k} + 1$ to compute the MAC. If there is a replay attack, then the receiver will test $C^L_{j,k} = C^L$ to be true and use $C^M_{j,k} + 1$ 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

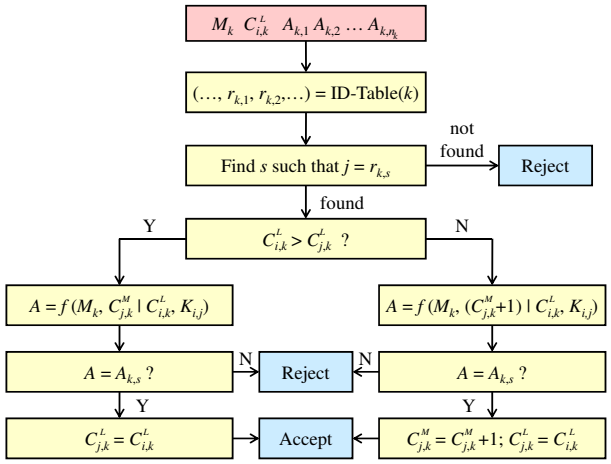

Fig. 3. The steps performed by a receiver N_j of a message M_k sent by a sender N_i.

P	Q					
	10^{-1}		10^{-4}		10^{-7}	
	Load	Avg L.	Load	Avg L.	Load	Avg L.
10^{-1}	1.0094	1.0241	1.0113	1.0267	1.0131	1.0288
10^{-2}	1.0150	1.0322	1.0169	1.0394	1.0188	1.0425
10^{-3}	1.0206	1.0445	1.0225	1.0481	1.0244	1.0506
10^{-4}	1.0282	1.0591	1.0300	1.0625	1.0319	1.0646
10^{-5}	1.0338	1.0668	1.0357	1.0733	1.0375	1.0767
10^{-6}	1.0394	1.0789	1.0413	1.0832	1.0432	1.0883
10^{-7}	1.0469	1.0987	1.0488	1.1007	1.0507	1.1040
10^{-8}	1.0526	1.1061	1.0544	1.1129	1.0563	1.1181
10^{-9}	1.0582	1.1213	1.0601	1.1232	—	—
10^{-10}	—	—	—	—	—	—

TABLE III
The relative bus load and average message latency under $n_k = 1$ and different values of P and Q where "—" means that there is no feasible solution. Without the security mechanism, the original bus load 376.44kbps and average message latency 11.535ms are both scaled to 1.

P	Q					
	10^{-1}		10^{-4}		10^{-7}	
	Load	Avg L.	Load	Avg L.	Load	Avg L.
10^{-1}	1.0244	1.0506	1.0263	1.0571	1.0282	1.0591
10^{-2}	1.0413	1.0832	1.0432	1.0883	1.0451	1.0968
10^{-3}	1.0582	1.1213	1.0601	1.1232	—	—
10^{-4}	—	—	—	—	—	—

TABLE IV
The relative bus load and average message latency under $n_k = 3$ and different values of P and Q where "—" means that there is no feasible solution. Without the security mechanism, the original bus load 376.44kbps and average message latency 11.535ms are both scaled to 1.

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 0.1^8. 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 address this potential issue by providing counter reset mechanisms, but they are not covered in this paper.

IV. Experimental Results

In this section, 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:

- q: the probability that a message is missing due to a network fault.
- R: the bus data rate.

The following message M_k parameters are defined:

- n_k: the number of the message receivers.
- P_k: the upper-bound of the probability of a successful attack.
- Q_k: the upper-bound of the probability that a counter is out of synchronization.

If M_k is not a security-critical message, then $P_k = Q_k = 1$.

Since there is no global time in CAN, the approaches in [4], [5], [6], [8], [9], [10] 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). Table III and Table IV show the relative bus load and average latency (computed by [3]) with different values of P and Q, where $P_k = P$ and $Q_k = Q$ for all k, under the assumptions that the n_k 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 security mechanism impacts on the system bus load and message latency. If there exist tight constraints on the bus load, the average message latency, or the message latency (deadline) for each message, then we can check if the security mechanism can be applied or not. As shown in Table III, when $n_k = 1$, if we want to make sure that the probability of a successful attack and the probability that a node is out of synchronization are both bound by 10^{-4}, then there is a 3% increase on the bus load and a 6.25% increase on the average message latency. Note that, in some cases where the values of P and Q are both large, there is no feasible solution. For our experiments, we show that we can achieve a very high security level (*e.g.*, $P(\text{successful attack}) \leq 10^{-8}$), with a bus load or average message latency increasing less than 6% and 12%, respectively. However, as shown in Table IV, when $n_k = 3$, we can see that the feasible region is reduced, since there are fewer bits available.

V. Conclusions

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.

Acknowledgements

We would like to thank Paolo Giusto, Joseph D'Ambrosio, Dave Nairn, and Tom Forest from General Motors for the valuable discussions and feedback.

References

[1] Controller Area Network, Specification 2.0.

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

[3] M. Di Natale, H. Zeng, P. Giusto, A. Ghosal, "Worst-case time analysis of CAN messages," Understanding and Using the Controller Area Network Communication Protocol, Springer, pp. 43–65, 2012.

[4] A. Perrig, R. Canetti, D. Song, and D. Tygar, "Efficient authentication and signing of multicast streams over lossy channels," IEEE Symposium on Security and Privacy, 2000.

[5] A. Perrig, R. Canetti, D. Song, and D. Tygar, "Efficient and secure source authentication for multicast," Network and Distributed System Security Symposium, 2001.

[6] A. Perrig, R. Canetti, D. Song, and D. Tygar, "The TESLA broadcast authentication protocol," RSA Cryptobytes, 2002

[7] S. Shenker and I. Stoica, "Security," Slides of Lectures 13 and 14, CS 194: Distributed Systems, University of California, Berkeley, 2005

[8] C. Szilagyi and P. Koopman, "A flexible approach to embedded network multicast authentication," Workshop on Embedded System Security, 2008.

[9] C. Szilagyi and P. Koopman, "Flexible multicast authentication for time-triggered embedded control network applications," International Conference on Dependable Systems and Networks, 2009

[10] C. Szilagyi and P. Koopman, "Low cost multicast authentication via validity voting in time-triggered embedded control networks," Workshop on Embedded System Security, 2010.

[11] M. Wolf, A. Weimerskirch, and C. Paar, "Security in automotive bus systems," Workshop on Embedded Security in Cars, 2004.