LCAP - A Lightweight CAN Authentication Protocol for Securing In-Vehicle Networks

(Full Paper)

Ahmed Hazem*
Valeo Egypt
Smart Village, Giza, Egypt
Email: ahmed.hazem@valeo.com

Hossam A. H. Fahmy
Electronics and Communications Engineering Dept.,
Cairo University
Giza, Egypt
Email: hfahmy@alumni.stanford.edu

***Abstract*—The design of in-vehicle communication networks has been always concerned with reliability and safety. There was no much attention paid to the security of such networks. This is because there was no clear evidence if the security of such networks could be compromised. However, recent experiments have shown the consequences of compromising in-vehicle networks. Those experiments relied on vulnerabilities that exist in such networks. Some of those vulnerabilities arise from the used communication standards themselves. Therefore, it became necessary to find ways of introducing security to the used standards. In this paper, a lightweight message source authentication protocol for Controller Area Network (CAN) [1] is proposed and fully implemented. The main advantage of the protocol is that it can be practically deployed in vehicles manufactured nowadays with minimum overhead. The protocol does not require any hardware modifications to be done in the CAN network. Also, it does not add much overhead to the embedded software of the ECUs. Moreover, it tends to avoid any modifications from being done to the existing CAN message sets that are being exchanged between ECUs.**

I. Introduction

Automotive industry has been affected dramatically by the advances of electronics during the last century. The introduction of electronics into cars has made them no more pure mechanical systems. Almost every new car manufactured nowadays contains tens of electronic control units (ECUs) [2]. ECUs have been introduced initially in cars for the purpose of engine management. Later on, they have been used for controlling many systems inside cars like brakes, transmission, airbags, climate control, power windows, infotainment and telematics. Also, they have been used to add new capabilities to cars like parking assistance, lane departure warning, blind spot detection, etc.

Electronic control units themselves are not pure hardware components. Instead, they consist of both software and hardware. In most cases, the ECU consists of a set of electronic circuits that are controlled using a microcontroller running from tens to hundreds thousands lines of code. As a result, a considerable amount of production defects may arise from software. When such defects are discovered after cars have been sold, the ECUs need not to be physically replaced in order to fix those defects. Typically, reprogramming the software of the ECU can solve everything. If the defects are not critical, then car manufacturers do not need to recall their defected cars in order to fix them. Instead, software reprogramming can be made during regular service that is made in authorized workshops.

* This work has been done as a part of M.Sc. thesis prepared at the Electronics and Communications Engineering Dept., Cairo University.

A. New threats

Although it sounds good to have the ability to fix defected parts using software, it introduces many safety considerations. Consider an attacker who has physical access to an ECU and could insert malicious code into it. Since most of the car internal networks are linked together, it is possible for this malicious code to control several safety-critical parts of the car. This highlights a major risk which has been demonstrated in [3]. However, if we assume that the attacker may have physical access to the car then he may be able to replace complete ECUs not only replace the software flashed in them.

During the development of communication networks inside cars, it was always assumed that the network is a trusted zone. May be this assumption came from the fact that cars had limited interfaces with the outside world. The On-Board Diagnostics (OBD II) port was the main external interface between the car network and the outside world. Recently, cars contain several wireless interfaces for many purposes. For example, cars now contain tire pressure monitoring systems (TPMS), smartphone integration using bluetooth, web connectivity, etc. Other new interfaces may be deployed for vehicle-to-vehicle and vehicle-to-infrastructure communications. Thus, cars are no more closed systems. Accordingly, the car network is no more a trusted zone. As a result, it is required to secure the internal network of cars from the threats that may arise from the new interfaces. This security may be needed either at the interfaces themselves or within the in-vehicle network or both.

B. Experimental analysis of attacks

All the threats discussed above have always been treated as impractical until Koscher et al [3] demonstrated how an attacker can control various car functionalities - including safety critical systems - ignoring user input. The analysis was made in two parts. The first part used direct physical access to the vehicle network through the OBD II port. The second part [4] investigated using several indirect wireless access methods.

1) Direct Access: In [3], direct physical access to the car network was feasible through the OBD II port. Having such access, it was possible to listen to all messages exchanged on the Controller Area Network (CAN) bus. Also, it was possible to inject messages into the network. Moreover, it was possible to dump the code of some ECUs using ReadMemorybyAddress diagnostic service. Using that method of attack it was possible to control several systems of the car like the engine, brakes, body controller, instrument panel cluster, radio and HVAC.

2) Indirect access: In [4], several experiments have been done using indirect access means. In the first method, the adversary could gain access to the PassThru device that connects the workshop computer to the car under service. Using this access, it is possible to spread an attack among all cars that are being serviced. Another method of indirect access is the infotainment systems. Modern infotainment systems are connected to the in-vehicle networks as well as other external interfaces. Those interfaces include iPhone integration, web connectivity. It is clear that compromising any of those interfaces could cause threat to the in-vehicle network. In [4], it was demonstrated that the firmware of the infotainment system of the car under test could be updated by inserting a CD-Rom with a certain image name into the CD-Player.

C. Reasons of vulnerabilities

1) Inherent weaknesses: In [3], it has been highlighted that some of the weaknesses of vehicles security arise from the properties of the CAN bus itself such as:

- Broadcast Nature
  Since the bus has a broadcast nature, then any node connected to the bus can listen to all data exchanged.
- Fragility to Denial of Service (DoS)
  Based on the arbitration scheme of CAN, any node can put the bus in a dominant state preventing other nodes from sending any messages.
- Absence of authentication
  The CAN message itself does not contain any authentication information about its sender. Thus, it is possible for any attacker who connects to the bus to send messages using the identity of any trusted node.

2) Weaknesses due to deviation from standards: In addition to the weaknesses mentioned above, there are also some weaknesses that arise from deviating from security standards and regulations.

- Reflashing the ECU should be allowed when the car is stationary only. However, it was possible to reflash some ECUs in a moving car.
- The key used for reflashing ECUs should be different from an ECU to another. However, it was found that sometimes all the cars have the same key for a certain ECU.
- Standards specify that a CAN gateway can be reflashed only from the high speed network; and not from the low speed network. However, it has been discovered that a gateway can be also reflashed from the low speed network.
- The stored keys in each ECU shall have restricted access so that they cannot be disclosed easily. However, it was possible to retrieve some keys from ECUs.

D. Impact of attacks

The highlighted weaknesses in the security of in-vehicle networks could attract the attention of criminals. Basically, thieves may be able to locate cars, unlock the doors and steal them. It could be also possible to extend the attacks to include several cars in a city simultaneously causing severe accidents.

E. Motivation of the paper

Based on the risks highlighted above, it is important to find ways to enhance the security of vehicles. The paper focuses on one of the major security requirements for in-vehicle networks which is message source authentication. The paper is organized as follows. The next section shows the related work that has been published. Then, a lightweight authentication protocol is proposed to be used for CAN networks. After that, we analyze the protocol and compare it with other protocols.

II. Related Work

In this section, we discuss the related work that has been published in the field of in-vehicle security and multicast authentication in general.

A. EVITA Project

One of the largest projects that aimed at securing in-vehicle communication networks is the EVITA project [5]. The goal of the project was to provide a basis for the secure deployment of electronic safety aids based on vehicle-to-vehicle and vehicle-to-infrastructure communication. The target was to complement other e-safety related projects that focus on protecting the communication of vehicles with the outside by focusing on on-board network protection.

This section discusses some of the work done within that project.

1) Security requirements: The EVITA project has inferred the a set of security requirements and related functional requirements in order to satisfy the stated security objectives [6]. The requirements include: integrity / authenticity of e-safety related data, secure execution environment, vehicular access control, trusted on-board platform, secure in-vehicle data storage, confidentiality of certain on-board and external communication, privacy and interference of security functionality.

2) Security Module: In [7], Marko Wolf et al introduced the idea of using a *security module* for providing different cryptographic functionalities to vehicles. Both centralized and distributed approaches were discussed. In the centralized approach, a single security module was used for providing security to several ECUs within the car. On the other hand, the distributed approach was based on attaching a security module to each ECU that needs protection. From implementation point of view, both software and hardware can be used for realizing such security module. In the case of centralized approach, the hardware implementation is more suitable, while in the case of distributed approach, the software implementation is more practical.

3) Key Distribution Protocol for CAN: In [8], a key distribution protocol has been introduced for securing in-vehicle communications over CAN bus. In each ECU, a hardware security module (HSM) was attached. The HSM implements some cryptographic primitives as well as securing the key storage. The exchange of shared keys is done through a logical entity called "Key Master" (KM). Each node, has two keys to communicate with the KM; one for authentication and the other for transporting generated keys. To establish a secure communication channel between an ECU and other ECUs, the following steps are followed:

1) The ECU generates a pair of keys; one for generation and the other for verification.
2) The ECU sends the verification key encrypted to the KM.
3) The KM forwards the key encrypted to all other ECUs.

Those generated session keys are made valid for a limited time only. It is valid for one drive cycle for at most 48 hours. Segmentation of data had to be used by enhancing the standard ISO 15765-2 [9]. The proposed length of message authentication code (MAC) is only 32-bits. This is due to the low speed of the bus as well as the high load.

B. Securing the CAN bus

1) Message Authentication Protocol over CAN: The problems associated with implementing a backward compatible message authentication protocol on the CAN bus has been discussed in [10]. The following authentication protocol requirements were addressed: Message authentication, Replay attack resistance and Backward compatibility. The first requirement was met by attaching a Message Authentication Code (MAC) to a message. The algorithm proposed in the paper was Hash-based Message Authentication Code (HMAC) [11]. The second requirement was met by inserting a counter value inside MAC calculations. The most challenging requirement was the last one. This was because adding any extra data to a message would exceed the maximum possible length of a message (8 bytes). The proposed solution for use was to use an out-of-band protocol like CAN+ [12].

Using CAN+, additional data bits can be inserted within the transmission period of each CAN bit. The length of authentication data that was sent using CAN+ was 15 bytes (120 bits). Those 120 bits are divided into 2 parts; 8 Bits for carrying the status and the remaining 112 bits for carrying the payload. The authentication data consists of counter value and a signature. The counter value is used to prevent replay attacks. A node accepts a message when the received counter value is greater than the last value. When the counter value is about to saturate, a new session key has to be established. The handling of unauthorized messages uses the same error mechanism used by regular CAN nodes.

2) CAN message encryption using AES and attacking it using CPA: In [13], confidentiality has been added to CAN using AES symmetric encryption. The breaking of this algorithm using side-channel analysis has been studied as well. The parts of CAN messages that was chosen to be encrypted were the ID (11-bits), DLC (4-bits) and the data (up to 8 bytes). Thus the length of plaintext was 10 bytes. However, the block size of AES is 128 bits (16 bytes). Hence, padding was added to the 10 bytes of plaintext before the encryption is done. The resultant of encryption is also 16 bytes, thus requiring two CAN messages to send it. Hence, two message IDs are allocated for transmitting these encrypted messages.

3) Multiple MAC per receiver: Using symmetric cryptography for multicast authentication has been also discussed in [14]. In this paper, the sender creates multiple MACs for each message. Each MAC is calculated using a key that is shared between the sender and one of the receivers. The sender appends all those MACs to the message being transmitted. Each receiver uses its key to verify part of the MAC. The papers are concerned with multicast authentication for automotive networks including CAN, FlexRay and Time-Triggered Protocol. Concerning CAN, it is proposed to use only half of the payload of each CAN message for carrying data, while using the remaining 4 bytes for carrying MACs. If each MAC is composed of one byte, then the message can carry 4 MACs corresponding to 4 receivers only.

C. The TESLA protocol and its variations

TESLA protocol was proposed in [15] as a new protocol for multicast authentication. The main idea behind it is to achieve asymmetric properties by using delayed disclosure of keys. However, this leads to delayed authentication. This delayed authentication has two main drawbacks. First, the receiver need to have storage for some unauthenticated messages till their key is disclosed. This increases the effect of DoS attack where an attacker may flood the receiver with many wrong messages. The second point is that this makes TESLA not suitable for realtime systems. The protocol was published later as RFC [16].

In order to achieve immediate authentication, a modification to TESLA protocol was proposed in [17]. In the proposal, messages do not need to be queued at the receiver waiting for being authorized. Instead, they are queued at the sender putting the hash code of each message in the preceding one. This solved the problem of DoS attack. Later on, μTESLA [18] was developed in order to be used in wireless sensor networks. The main aspects of such systems is the lack of processing capabilities, low memory for storing code, small RAM and running on battery power devices. TESLA was also modified to be used in Secure Real-time Transport Protocol (SRTP) as in [19]. Recently, another modification was made for TESLA introducing TESLA++ [20]. TESLA++ was developed to be used in Vehicular Ad-hoc Networks (VANETs). It modifies TESLA in a way that makes it resilient to memory-based DoS attacks.

III. Lightweight CAN Authentication Protocol

In this section, we propose an authentication protocol to be used for securing CAN networks.

A. *Threat Model and Security Requirements*

In-vehicle networks consist of various ECUs that are interconnected using communication buses. There are many types of communications buses used like CAN [1], LIN [21], FlexRay [22] and MOST [23]. CAN is the most used type of buses. As discussed earlier, CAN messages do not contain any source or destination addresses. Also, it does not provide any means of authentication. Hence, any adversary node that succeeds to gain access to the bus can listen to any transmitted message. Moreover, it can inject malicious messages into the network.

In this section, we propose a new authentication protocol that can be deployed inside in-vehicle networks. The protocol is designed mainly to be used inside CAN networks. However, it can be also used in other communication bus types.

When protecting in-vehicle network from a compromised node, we consider two cases:

Case A: The CAN bus has some ECUs connected to it. One of these ECUs was previously compromised by an adversary.

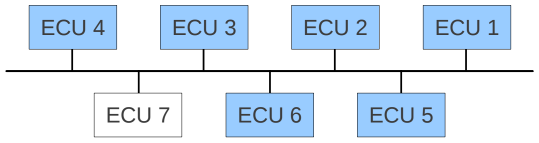

Fig. 1. Case A: Existing ECU is compromised

Case B: The CAN bus has some ECUs connected to it. All ECUs are communicating correctly. The adversary node is attached to the CAN bus at some point of time. According to the CAN bus specification, the attached node can listen to all exchanged CAN messages. Also, it has the ability to send any CAN message on behalf of any other node.

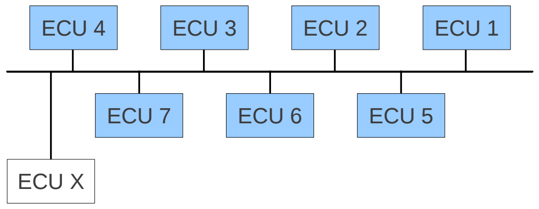

Fig. 2. Case B: Adversary node connected to the bus

For both cases, in order to protect the network against such attacks, message source authentication is required. However, the CAN bus protocol does not specify any means of authentication. As a result, it is required to design a higher level authentication protocol that can be adopted in automotive CAN networks. There are 2 main challenges when designing that authentication protocol.

1) The protocol shall add a small communication overhead. The payload of any CAN message is already too small (8 bytes by maximum). For the current networks, those 8 bytes are highly utilized. Thus, the smaller the overhead used, the easier to deploy the authentication protocol in the currently designed networks with minimum reformatting of messages.
2) The protocol shall not require either heavy computation or high memory consumption. This is because the currently produced ECUs use microcontrollers with limited resources.

B. *The CAN Authentication Protocol*

1) Mutlicast Authentication Overview: Basically, in order to achieve message source authentication we need to satisfy the following requirements:

- The message shall contain an evidence that can be generated only by its trusted sender.
- The receiver shall be able to verify that evidence.
- The receiver shall not be able to re-transmit the message masquerading the trusted sender.

In the case of unicast communication (which is not the case of CAN bus), symmetric cryptography can be used to achieve authentication between the two communicating entities. The two entities have a shared secret that can be used to prove the authenticity of the sender.

However, in the case of multicast communication (which is the case of CAN bus), asymmetric cryptography is needed in order to prevent any receiver from sending messages on behalf of the original sender. Unfortunately, asymmetric cryptography is too heavy to use in our domain specially when implementing it in software.

If we try to use TESLA in CAN networks we will find that we will need large communication overhead because each message shall contain the original data to be transmitted in addition to the MAC and a key. Also, the delay introduced by TESLA in unacceptable for in-vehicle networks as real-time systems.

2) Main Idea: From our point of view, it is only required to append a magic number to the message that can be verified by the receiver. This magic number can be only selected by the sender and verified by the receiver. The process of generating this magic number is based on the one-way hash function employed in TESLA protocol. The sender selects a random number then applies a transformation function multiple times. The result is used in reverse order. The last generated value of the chain is communicated initially to each receiver. Each receiver can verify the message by applying the transformation function on the current received value and compare it to the previous value. Keeping in mind that the payload of a CAN message is only 8 bytes, then we should not add a large overhead. The proposed length of the magic number is 2 bytes.

3) Modes of Operation: The protocol can be used in one of two modes:

- Extended Mode
- Standard Mode

In the *Extended Mode*, the magic number is sent using the "Extended Identifier" field of the CAN message. Thus, the payload of the original CAN message is not affected. This requires that extended identifier is enabled in the CAN controller, but all its bits are masked in order to receive all messages and then apply authentication in the upper levels. This mode is suitable for messages having standard identifier only.

In the *Standard Mode*, 2 bytes of the CAN message payload are used for sending the magic number. For CAN messages whose payload is less than or equal to 6 bytes, this does not add any overhead. However, for larger messages, re-formating is needed for the messages as 75% of the message is only usable. This mode is suitable for messages having extended identifier where the Extended Mode described above cannot be used.

In order to increase the security of the protocol, the payload of the CAN message including the magic number shall be encrypted using a symmetric key. The shared key used in encryption shall be communicated to each receiver. Note that this encryption increases the security of case B only. This is because in case A, one of the receivers is compromised and hence knows the encryption key.

C. Protocol Details

1) Assumptions: In a CAN network, there exists multiple nodes which are communicating together in a broadcast way. In our analysis, we consider a set of n sender nodes $S_1, S_2, ...S_n$ that are broadcasting p messages to m receiver nodes $R_1, R_2, ...R_m$. For each pair of sender-receiver, there exists a shared secret that is stored in both ECUs. It is assumed that it is stored in a protected memory that cannot be easily read. When an ECU is replaced, it shall be calibrated with other existing ECUs so as to set communication keys correctly.

2) Notation:

- The symbol '|' is used to concatenate bytes together.
- i: The index of the message to be transmitted, where $i \in (1, p)$.
- j: The order of the message to be transmitted, where $j \in (1, \infty)$.
- D_{ij}: The data to be transmitted by the j^{th} message of index i.
- M_{ij}: The magic number transmitted with the j^{th} message of index i.
- K_S: The session key.
- $E(x, k)$: Encryption function that encrypts x using the key k.
- $D(x, k)$: Decryption function that decrypts x using the key k.
- $H(x, k)$: HMAC function applied on a variable x using the key k.

3) Protocol Parameters:

- λ : The size of the magic number chain.
- α: The length of the magic number in bits.
- δ: The maximum number of trials made by the receiver to detect lost messages.
- τ: The time after which a sender is considered absent.

The parameter λ depends on two main things. It should be less than 2^α in order not to repeat the magic number. Also, it specifies the memory requirements for the sender. The larger the value of λ, the larger memory is used by the sender to store the chain. If λ takes its maximum possible value 65535, then 128 kilobytes are needed to store the chain. According to the capabilities of ECUs used in today's vehicles, this memory size is hard to achieve.

4) Handshaking: During various phases of the protocol, many handshaking messages are exchanged between senders and receivers. This requires defining new CAN messages to the network in which the authentication protocol is going to be deployed. All the messages have the same standard CAN identifier (11 bits), but they differ in the value of the extended identifier (18 bits). For each pair of a sender and a receiver, five messages are defined:

- Channel Setup Request
- First Response Message
- Consecutive Response Message
- Soft Sync Request
- Hard Sync Request

Thus, the total number of needed CAN message identifiers to be added is equal to $5\times$ Number of senders $\times$ Number of receivers. The format of different handshaking messages is shown in figures 3, 4 and 5.

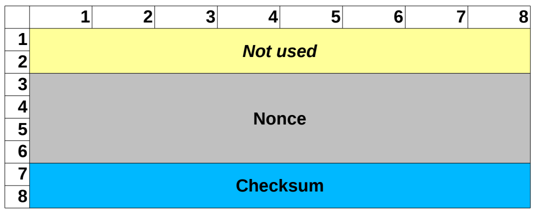

Fig. 3. (Channel Setup) / (Soft Sync) / (Hard Sync) Request Message

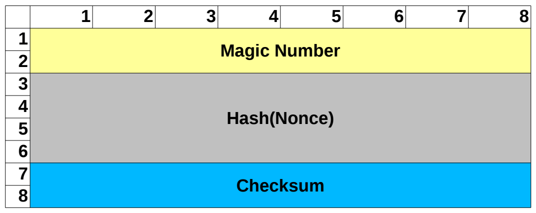

Fig. 4. First Response Message

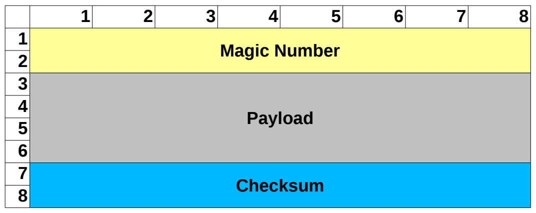

Fig. 5. Consecutive Response Message

As shown in the figures 3, 4 and 5, the last 2 bytes of each message are used to contain the checksum of the first 6 bytes. This is to preserve the integrity of the message. For "First Response" and "Consecutive Response" messages, the 1^{st} 2 bytes of the message contain a "magic number" that is used to authenticate the messages.

D. Protocol Phases

The protocol consists of the following phases:

- Initialization
- Channel Setup
- Message Setup
- Data Exchange
- Chain Refresh
  For the protocol to be robust, two additional phases are needed:
- Soft Synchronization
- Hard Synchronization

1) Initialization: In this phase, each sender creates the "Handshaking Magic Number Chain", "Channel Magic Number Chain", the "Session Key" K_S and the "HMAC Key" K_H. Also, it generates the "Magic Number Chain" for each of the messages it transmits.

2) Channel Setup: In this phase, the sender distributes "Session Key", "Channel Initial Magic Number" and "HMAC Key" to each receiver separately. The sender encrypts this information using a symmetric key. For each receiver, it uses a separate key that is pre-shared between the sender and the receiver. This pre-shared key is programmed in the ECUs during production and shall be updated when an ECU is replaced. The length of this key is 128 bits.

The "Session Key" that is sent in this phase is the key that will be used later to encrypt/decrypt any data exchanged with the sender. The length of this key is chosen be 80 bits. Since this length does not fit in one message, then it will be sent in three parts. The "Channel Initial Magic Number" is a magic number to let the receivers authenticate the sender during "Message Setup" phase. The "HMAC Key" is the key that is used to perform HMAC operation during other phases. The length of the "HMAC key" is 16 bits.

The following steps are repeated for each receiver:

1) The receiver sends "Channel Setup Request" message to the sender. The message contains a 32-bit nonce. The message is encrypted using the pre-shared key between the two nodes.
2) The sender replies by a "First Response" message. The message contains the hash value of the nonce (truncated to 16 bits). It contains also a "magic number" that will be used to authenticate the sender during the rest of this phase.
3) The sender sends a "Consecutive Response" message containing the 1^{st} 4 bytes of the "Session Key". The "magic number" that is sent in this message can be authenticated by applying hash function on it and comparing it to the previously sent "magic number".
4) The sender repeats the previous step for the 2^{nd} and 3^{rd} parts of the "Session Key". For the 3^{rd} part, only two bytes of the payload are used.
5) Finally, the sender sends the "Channel Initial Magic Number" M_{c0} and the "HMAC key" K_H.

Note that steps from 1 to 4 are encrypted using the pre-shared key of the 2 ECUs while step 5 is encrypted using the session key K_S.

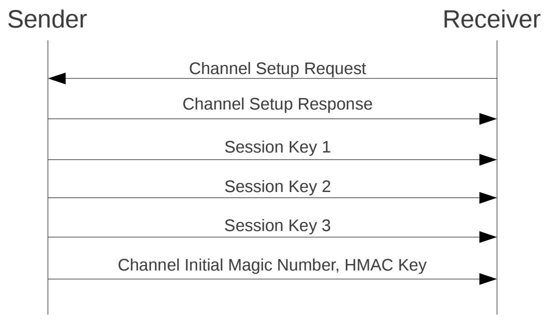

Fig. 6. Channel Setup

When the receiver does not receive response from a sender after a certain timeout period defined by the parameter τ, then the receiver shall consider this sender absent from the network. Finally, at the end of this phase, each receiver will have the "Session Key", the "HMAC Key" and the "Channel Initial Magic Number". A sender shall respond to "Channel Setup" only at the start of a new driving cycle. This is to protect the sender against denial of service attacks.

Strength of the pre-shared key Since the length of the pre-shared key is 128 bits, then it needs 2^{127} trials on average in order to break the key. The time needed for a trial is bounded by the minimum of the time needed for initiating a new driving cycle. Assuming that this value can be as low as 1 second. Then the time needed to make the 2^{127} trials is 5.4×10^{30} years.

Strength against replay attacks Replaying a receiver request to the sender may allow the sender to do the channel setup based on an invalid nonce value. In this case, the receiver will not accept any of the sent parameters. Replaying sender responses is not possible because it depends initially on the value of the nonce and later on, it depends on the values of the magic numbers. In both cases, the receiver will reject the replayed value.

Strength of the HMAC key The length of the "HMAC Key" is chosen to obtain HMAC security of 16 bits. According to [24], the security of HMAC is divided into two parts; the security of the HMAC algorithm and the security of the HMAC value. The security of the HMAC algorithm is the minimum of the security of HMAC key and twice the length of the output of the used hash function. The latter parameter is 32 bits. Therefore, from this point of view, the length of HMAC key shall be set to 32 bits. On the other hand, the security of the HMAC value is bounded by the length of the HMAC output which is 16 bits. Therefore, it is sufficient for the length of the HMAC key to be 16 bits.

3) Message Setup: In this phase, the sender sends the initial magic number of each message that it transmits to all its receivers. This is done in a broadcast way, i.e. it sends the initial magic number of each message to all receiving nodes; not to each node separately. This information is sent using the same data message; not the handshake message. The payload of the data message in this case consists of the following:

- Magic Number (That can be authenticated using the "Channel Initial Magic Number" that has been sent during the "Channel Setup" phase).
- Initial magic number for the data message.

Note that the sender cannot use the same magic number to authenticate all messages that it sends. Therefore, the messages shall be ordered in a way such that each message uses an order of the magic number. For example, the first message can be verified by applying the one-way hash function once, while the second message can be verified by applying the one-way hash function twice and so on.

Strength of the session key The size of the session key is 80 bits. Then, for a brute force attack it needs 2^{79} trials on average in order to determine the key. However, these trials have to be done during the validity period of the session key. As mentioned earlier, the session key lasts for a complete driving cycle - which is assumed - to take 24 hours by maximum. Then, it is required to do $2^{79} = 6 \times 10^{23}$ trials in 24 hours in order to break the session key. This means that it is required to do 6.9×10^{12} trials per microsecond which is impractical to do with the speed of CAN.

4) Data Exchange: Once the "Session Key" K_S, "HMAC Key" K_H and the "Initial Magic Number" M_{i0} of each CAN message are sent to all receivers, the data exchange can begin.

Sender The sender uses the magic number chain in a reverse order.

1) The sender appends the current magic number M_{ij} to the current message D_{ij}
2) The sender encrypts the resultant using the session key K_S

The sent message takes the form:

$$E((M_{ij}|D_{ij}), K_s)$$

In "Standard Mode", the result of encryption in the equation above is sent in the payload of the CAN message. However in "Extended Mode", the first 2 bytes are sent using the extended identifier field, while the rest is sent in the normal payload of the CAN message.

Receivers The receiver verifies the message by the following steps:

1) The receiver decrypts the message using the "Session Key" K_S.
2) The receiver extracts the magic number M_{ij}.
3) The receiver verifies the magic number by applying the HMAC function on it (using the HMAC key K_H), truncating the result and comparing it to the previous magic number.

5) Chain Refresh: It is required to refresh each magic number chain used periodically. A separate chain refresh phase is required. Before the current used chain expires, the sender uses the current authenticated channel to transmit the new initial magic number to all receivers.

The steps taken by the sender are as follows:

1) When sending the message number $\lambda - 1$, the sender does not send the regular data message, but sends the new initial magic number of the chain. This message is authenticated using the last element in the current magic number chain.
2) All receivers will receive the new initial magic number.

Refreshing the magic number by this way has a drawback that a regular data message is dropped in order to send the new initial magic number. This can be acceptable for some messages while it cannot be accepted for others. For the latter case, an out-of-bound message (using a separate CAN identifier) shall be used for the purpose of refreshing the chain.

Security of the chain length According to [24], the pre-image strength of a truncated hash function is equal to the truncated length (16 bits in our case). In our implementation, we chose λ to take the value of 100 which is much less than 2^{16}.

6) Soft Synchronization: At any point of time, any of the receivers may lose synchronization and want to synchronize the values of the magic numbers of all the messages it receives. In this case, the receiver uses the following sequence with all its senders:

1) The receiver R sends sync request message to the sender S.
2) The sender S replies by the current magic number for each message that it sends to that receiver (encrypted by the session key)

The authentication of exchanged messages is done in the same way as in "Channel Setup" phase.

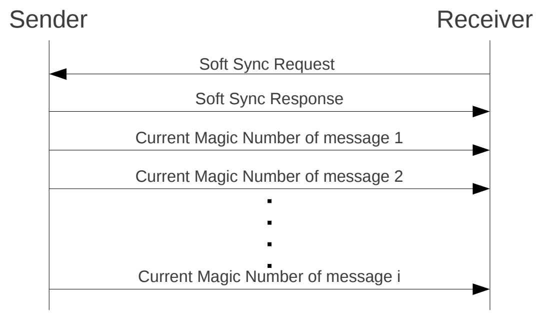
Sender
Receiver
Soft Sync Request
Soft Sync Response
Current Magic Number of message 1
Current Magic Number of message 2
Current Magic Number of message i

Fig. 7. Soft Synchronization

7) Hard Synchronization: If any of the receivers loses the "Session Key" or the "HMAC Key", then it needs to perform hard synchronization. In this case, the following sequence is used:

1) The receiver R sends a hard synchronization request to the sender S.

2) The sender S replies with the “Session Key” and the “HMAC Key”.
3) The sender sends current magic numbers in the same way as in soft synchronization.

The authentication of exchanged messages is done in the same way as in “Channel Setup” phase.

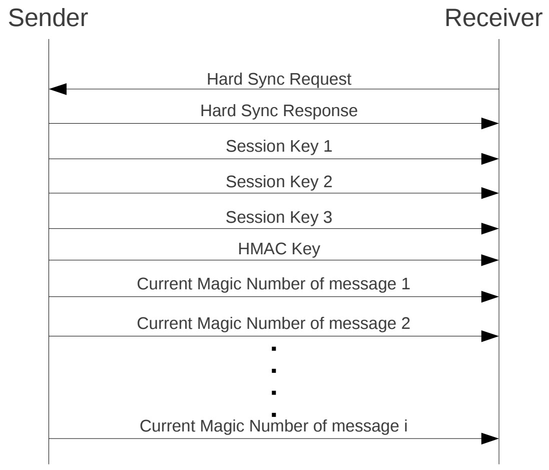
Sender
Receiver
Hard Sync Request
Hard Sync Response
Session Key 1
Session Key 2
Session Key 3
HMAC Key
Current Magic Number of message 1
Current Magic Number of message 2
Current Magic Number of message i

Fig. 8. Hard Synchronization

E. Cryptography

As described above, the protocol needs some cryptographic functions to be used. There are 3 functions required which are:

- Encryption/Decryption
- One-Way Hash Function
- Random number generation

1) Encryption/Decryption: Encryption/Decryption is used for all exchanged messages. The maximum size of data to be encrypted is the maximum size of a CAN message in addition to the size of the extended identifier. The size of the extended identifier is 18 bits, but only 16 bits are used for sending the magic number in “Extended Mode”. Hence, the maximum size of data to be encrypted is 10 bytes. According to the assumptions above, the session key size is 10 bytes. Due to the small size of data, symmetric stream cipher is used. Any stream cipher can be used. It is recommended to use any stream cipher recommended by eStream project [25]. However, when the used cipher requires key size larger than 12 bytes, then the protocol has to be modified in either of two ways. Either to keep the same size of exchanged keys fixed, but add a predefined part of the key shared between all nodes. The other solution is to increase the number of messages exchanged in the starting phase. For simplicity, we use RC4 to implement the protocol.

2) One-Way Hash Function: One-Way hash function is used to generate the chain of the magic numbers. If we use a hash function like SHA-256 the resultant chain will be always the same for the same initial seed i.e. the chain would be static. Therefore, if an attacker could generate the chain offline, then he could be able to send messages on behalf of the authenticated sender. Therefore, we choose to use HMAC [11]. Then, we truncate the result of the HMAC function to 2 bytes only in order to fit for the chosen magic number size. According to [24], the minimum length of the truncated result shall be greater than 8-bits. In our case, we take the leftmost 16-bits.

3) Random Number Generation: Random number generation is used to generate the “Session Key”, “HMAC Key” and the magic number chains. In our implementation, we used “rand” function of the C library. This function needs a random seed to be used each time. This seed could be generated using any unused ADC channel of the microcontroller. However, it is recommended to use a stronger pseudo-random function rather than “rand”.

F. Implementation

The authentication protocol was implemented on Starter-TRAK TRK-MPC5604B board manufactured by Freescale Semiconductors. The board contains MPC5604B PowerPC microcontroller. The main features of the microcontroller are:

- CPU: e200Z0h
- Max clock frequency: 64 MHz
- Code Flash: 512 KB, Data Flash: 64 KB
- Number of CAN controllers: 3

The protocol was implemented in C-code running at 64 MHz.

IV. Analysis and Results

The main concern while designing the protocol was to make it simple, practical and lightweight so that its adoption inside automotive CAN networks is easy and with the minimum overhead and cost. In this section, we analyze the protocol and compare it to other protocols.

The proposed threat model considered two cases; case A and case B. Using our authentication protocol for case B, if the adversary node tries to send a message on behalf of its original sender, the receivers will not authenticate it. The same applies for case A as well. However, the data sent by the compromised node itself is not guaranteed to be correct although it is being authenticated.

In order to analyze the authentication protocol and compare it to other protocols, the following factors have to be taken into consideration:

- Hardware modifications
- Software overhead
- CAN message set modifications
- Response time
- Scalability
- Maintainability

A. Hardware modifications

The protocol does not need any hardware modifications to be done inside the CAN network. It works with traditional CAN transceivers and CAN controllers. This means that no additional hardware cost is needed to deploy it. From this point of view, the protocol is more practical to be deployed rather than the protocol proposed in [10] that needed new CAN controller and CAN transceiver. It is also better than using Hardware Security Module (HSM) defined in EVITA project [5] since the HSM needs additional cost to be added to each vehicle being manufactured. However, HSM provides stronger security features since it depends on hardware for implementing cryptographic functions.

B. Software and Response Time Overhead

This overhead can be divided into the following:

- CPU load and memory consumption needed for performing cryptographic calculations as well as the protocol logic.
- Response delay due to the time consumed in adding authentication data (at the sender side) and verifying it (at the receiver side).
- Initialization delay that is induced from the setup phases of the protocol.

The execution time of RC4 encryption / decryption of single CAN message took around 160μs. This is for extended mode; where 10 bytes are encrypted / decrypted. However, this is the time consumed in both generating the key stream and performing XOR operation. It is more efficient to create the stream once for each used key and use it many times. Also, it is recommended [26] to drop the first generated 512 bytes of the stream.

The execution time of HMAC using 16-bit key using different hash functions is shown in table 1. Initial results have been obtained for an unoptimized code. Then, the code was optimized so that part of the HMAC algorithm is performed once for each key, then the rest of the algorithm is performed for every subsequent run.

	MD5	SHA 1	SHA 224	SHA 256
Unoptimized Code	150	286	543	543
Optimized Code (1^{st} run)	144	278	539	540
Optimized Code (subsequent run)	86	154	285	285

TABLE I
EXECUTION TIME (IN MICROSECONDS) OF ONE HMAC OPERATION

HMAC calculation is needed to be performed at the receiver side for each received CAN message. According to the acceptance of how many messages that can be dropped, this operation can be needed many times per message.

In order to have a uniform load at the sender side when refreshing magic number chains, it is recommended that the sender generates the new chain while consuming the current chain. In other words, with every message that the sender transmits, it consumes an element of the current chain and at the same time, it generates a new element of the new chain.

C. CAN message sets modifications

One of the main points to consider when selecting an authentication protocol for CAN is its effect on the message sets within the CAN network in which the protocol shall be deployed. For “Extended Mode”, the CAN messages are not modified at all. For “Standard Mode”, only CAN messages whose payload is greater than 6 bytes shall be reformatted. The method described in [10] is better than our protocol in this point. This is because it uses out-of-band transmission and hence does not affect CAN message sets at all.

D. Scalability

In order to discuss the scalability, two parameters are studied. The first one is the number of exchanged CAN messages inside the CAN network. The other one is the number of ECUs that communicate together inside the network.

For the first parameter, the protocol timings are not affected with increasing the number of exchanged messages except for “Soft Synchronization” and “Hard Synchronization” where the time needed for such phases increases linearly with the increase of the number of messages. On the other hand, memory consumption increases linearly by increasing the number of messages since the sender has to generate a magic number chain for each sent message and keep it in the memory.

For the second parameter, the time consumed in “Channel Setup” phase increases with increasing the number of ECUs as follows. For each additional ECU that communicates with n existing ECUs, n channels need to be setup. This is a disadvantage of the protocol. However, its impact is minimized by choosing “Channel Setup” to be done at the initialization of ECUs only. It is also better than [14]; where a message authentication code (MAC) is added inside the payload of the message for each receiver. This consumes most of the payload and does not fit for large number of receivers.

E. Maintainability

When it is required to replace an ECU that implements the authentication protocol, calibration shall be done to the new ECU as well as other ECUs that communicate with it using the CAN bus. As described earlier, this step is needed to update the pre-shared keys that are used for the “Channel Setup” phase. As long as the calibration process is done in a secure way, then the protocol supports the maintainability of ECUs implementing the protocol.

V. FUTURE WORK

A. Reduce the time of Session Key distribution

The session key is distributed to each receiver separately. The disadvantage of this is that it consumes time to send the key to each receiver. Using a more efficient key distribution protocol is a challenging point. The challenge is in finding a protocol that does not need much computation so that it can be implemented in all ECUs connected to the CAN bus.

Using public-key cryptography to exchange keys can be an alternative. However, it requires that all ECUs be able to do heavy computations. Hence, the optimum solution can be to use a hybrid scheme; where ECUs with small computational capabilities exchange keys using the method described in this paper while other ECUs exchange keys using public-key cryptography.

B. Adapting the protocol to other communication standards

Designing an authentication protocol for the CAN bus is limited by the bandwidth available for transmitting authentication data. The upper bound for this bandwidth is 8 bytes assuming no traffic is sent on the bus. Therefore, it is recommended for in-vehicle networks to migrate to other buses offering larger bandwidth so that strong authentication is possible. The first candidate for migration is the FlexRay protocol which is already deployed in many cars. It offers better opportunity for authentication since the size of its payload can reach up to 254 bytes. The other strong candidate is One Pair Ethernet [27]. Historically, Ethernet was not used inside vehicles due to its bad performance in the electromagnetic environment inside the vehicle. However, with the introduction of One-Pair Ethernet, it became now possible to use Ethernet with message sizes up to 1500 bytes.

VI. Conclusion

The exposure of in-vehicle networks to the external world requires securing the communication inside these networks. However, security adds a cost represented by extra processing, memory, reformatting of messages and time delay - both at initialization and in runtime. At the same time, the small payload of CAN messages puts a limit on the strength of the security that could be added to the bus.

Some authentication protocols have been proposed before but their adoption was not easy because they either needed modifications in the physical layer of the CAN [10] or they added much overhead inside the CAN message [14] that reduced the size of the useful payload inside the message.

The main goal of the work presented in the paper is to add lightweight authentication to the CAN bus with the minimum impact on the currently deployed networks. The protocol shows good analysis results and proves that it is more practical than other published protocols and is low-cost as well. However, due to the small bandwidth available for exchanging authentication data, it is recommended to migrate in-vehicle communication networks to a higher bandwidth ones like FlexRay and One Pair Ethernet.

References

[1] *CAN Specification*, Robert BOSCH GmbH Std. Version 2.0, 1991.

[2] R. N. Charette, "This car runs on code," *IEEE Spectrum*, feb 2009.

[3] 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," in *Security and Privacy (SP), 2010 IEEE Symposium on*, Oakland, CA, USA, may 2010, pp. 447–462.

[4] 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*, Berkeley, CA, USA, aug 2011.

[5] E-safety vehicle intrusion protected applications (EVITA). Last accessed: July 2012. [Online]. Available: http://evita-project.org/

[6] L. Apvrille, R. El Khayari, O. Henniger, Y. Roudier, H. Schweppe, H. Seudié, B. Weyl, and M. Wolf, "Secure automotive on-board electronics network architecture," in *FISITA 2010, World Automotive Congress, 30 May-4 June 2010*, Budapest, Hungary, May/Jun. 2010.

[7] M. Wolf, A. Weimerskirch, and T. J. Wollinger, "State of the art: Embedding security in vehicles," *EURASIP Journal on Embedded Systems*, vol. 2007, 2007.

[8] H. Schweppe, Y. Roudier, B. Weyl, L. Apvrille, and D. Scheuermann, "Car2x communication: Securing the last meter - a cost-effective approach for ensuring trust in car2x applications using in-vehicle symmetric cryptography," in *Vehicular Technology Conference (VTC Fall), 2011 IEEE*, sep 2011, pp. 1–5.

[9] *Road vehicles – Diagnostic communication over Controller Area Network (DoCAN) Standard – Part 2: Transport protocol and network layer services*, ISO Std. 15 765-2, 2011.

[10] A. V. Herrewege, D. Singelee, and I. Verbauwhede, "CANAuth - a simple, backward compatible broadcast authentication protocol for CAN bus," in *9th Embedded Security in Cars Conference*, Dresden, Germany, nov 2011.

[11] *The Keyed-Hash Message Authentication Code (HMAC)*, National Institute of Standards and Technology Std. FIPS PUB 198-1, 2008.

[12] T. Ziermann, S. Wildermann, and J. Teich, "CAN+: A new backward-compatible controller area network (CAN) protocol with up to 16x higher data rates." in *Design, Automation Test in Europe Conference Exhibition, 2009. DATE '09.*, apr 2009, pp. 1088–1093.

[13] M. D. Hamilton, M. Tunstall, E. M. Popovici, and W. P. Marnane, "Side channel analysis of an automotive microprocessor," in *Signals and Systems Conference, 208. (ISSC 2008). IET Irish*, jun 2008, pp. 4–9.

[14] C. Szilagyi and P. Koopman, "A flexible approach to embedded network multicast authentication," in *3rd Workshop on Embedded Systems Security (WESS'2008)*, Atlanta, Georgia, USA, oct 2008.

[15] A. Perrig, R. Canetti, J. Tygar, and D. Song, "Efficient authentication and signing of multicast streams over lossy channels," in *Security and Privacy, 2000. S P 2000. Proceedings. 2000 IEEE Symposium on*, Berkeley, CA, USA, may 2000, pp. 56–73.

[16] A. Perrig, D. Song, R. Canetti, J. D. Tygar, and B. Briscoe, "Timed efficient stream loss-tolerant authentication TESLA: Multicast source authentication transform introduction," RFC 4082 (Informational), Internet Engineering Task Force, jun 2005. [Online]. Available: http://www.ietf.org/rfc/rfc4082.txt

[17] A. Perrig, R. Canetti, D. Song, and J. D. Tygar, "Efficient and secure source authentication for multicast," in *Network and Distributed System Security Symposium, NDSS '01*, San Diego, CA, USA, feb 2001, pp. 35–46.

[18] A. Perrig, R. Szewczyk, V. Wen, D. Culler, and J. D. Tygar, "SPINS: Security protocols for sensor networks," in *Seventh Annual International Conference on Mobile Computing and Networks (MobiCOM 2001)*, Rome, Italy, jul 2001.

[19] M. Baugher and E. Carrara, "The use of timed efficient stream loss-tolerant authentication (TESLA) in the secure real-time transport protocol SRTP," RFC 4383 (Proposed Standard), Internet Engineering Task Force, feb 2006. [Online]. Available: http://www.ietf.org/rfc/rfc4383.txt

[20] A. Studer, F. Bai, B. Bellur, and A. Perrig, "Flexible, extensible, and efficient VANET authentication," *Journal of Communications and Networks*, vol. 11, no. 6, pp. 574–588, dec 2009.

[21] *LIN Specification Package*, LIN Consortium Std. Revision 2.2A, 2010.

[22] *FlexRay Protocol Specification*, Flexray Consortium Std. Version 3.0.1, 2010.

[23] *MOST Specification*, MOST Cooperation Std. Rev. 3.0 E2, 2010.

[24] *Recommendation for Applications Using Approved Hash Algorithms*, National Institute of Standards and Technology, 2009.

[25] M. Robshaw and O. Billet, Eds., *New Stream Cipher Designs: The eSTREAM Finalists*. Springer, 2008.

[26] *ECRYPT II Yearly Report on Algorithms and Keysizes*, European Network of Excellence in Cryptology II, 2011.

[27] P. Hank, T. Suermann, and S. Mller, "Automotive ethernet, a holistic approach for a next generation in-vehicle networking standard," in *Advanced Microsystems for Automotive Applications 2012*. Springer Berlin Heidelberg, 2012, pp. 79–89.