SECURE MESSAGE AUTHENTICATION PROTOCOL FOR CAN (CONTROLLER AREA NETWORK)

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF APPLIED MATHEMATICS
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

SARP MERTOL

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
CRYPTOGRAPHY

JANUARY 2020

Approval of the thesis:

SECURE MESSAGE AUTHENTICATION PROTOCOL FOR CAN (CONTROLLER AREA NETWORK)

submitted by **SARP MERTOL** in partial fulfillment of the requirements for the degree of **Master of Science in Cryptography Department, Middle East Technical University** by,

Prof. Dr. Ömür Uğur
Director, Graduate School of **Applied Mathematics** _____

Prof. Dr. Ferruh Özbudak
Head of Department, **Cryptography** _____

Assoc. Prof. Dr. Ali Doğanaksoy
Supervisor, **Mathematics, METU** _____

Assoc. Prof. Dr. Fatih Sulak
Co-supervisor, **Mathematics, Atılım University** _____

Examining Committee Members:

Assoc. Prof. Dr. Murat Cenk
Cryptography, METU _____

Assoc. Prof. Dr. Ali Doğanaksoy
Mathematics, METU _____

Assoc. Prof. Dr. Oğuz Yayla
Mathematics, Hacettepe University _____

Date: _____

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name: SARP MERTOL

Signature :

ABSTRACT

SECURE MESSAGE AUTHENTICATION PROTOCOL FOR CAN
(CONTROLLER AREA NETWORK)

Mertol, Sarp
M.S., Department of Cryptography
Supervisor : Assoc. Prof. Dr. Ali Doğanaksoy
Co-Supervisor : Assoc. Prof. Dr. Fatih Sulak

January 2020, 34 pages

The widespread communication of vehicles with each other and road infrastructure has raised concerns about how to ensure network security of the electronic control units (ECUs) in the vehicle. The fact that networks such as the Controller Area Network (CAN), which is commonly used in in-vehicle communications, will also be connected to external networks (e.g. 3G / 4G mobile networks) will allow malicious adversaries to benefit from the vulnerability of the CAN. The authentication of messages of ECUs in the vehicle is required to ensure that in-vehicle communications are secured. However, the cryptographic algorithms and protocols that can be used for this message verification process should be selected considering the real-time communication requirement in the vehicle.

Keywords: Controller Area Network, Message Authentication Protocol, Message Authentication Code, Network Security, Replay Attack

ÖZ

CAN (CONTROLLER AREA NETWORK) İÇİN GÜVENLİ MESAJ DOĞRULAMA PROTOKOLÜ

Mertol, Sarp
Yüksek Lisans, Kriptografi Bölümü
Tez Yöneticisi : Doç. Dr. Ali Doğanaksoy
Ortak Tez Yöneticisi : Doç. Dr. Fatih Sulak

Ocak 2020, 34 sayfa

Araçların birbirleriyle ve karayolu altyapısı ile haberleşmesinin yaygınlaşacak olması, araç içerisinde bulunan birimlerin ağ güvenliğinin nasıl sağlanacağı ile ilgili endişeleri de beraberinde getirdi. Araç içi haberleşmede yaygın olarak kullanılan Denetleyici Alan Ağı (CAN) gibi ağların aynı zamanda harici ağlara (örn. 3G/4G mobil ağları) bağlanacak olması, kötü niyetli kullanıcıların CAN'in güvenlik konusundaki zayıflıklarından yaralanmasına olanak verecektir. Araç içi haberleşmenin güvenli hale gelmesi için araç içerisinde bulunan elektronik kontrol birimlerinin (ECU) mesajlarının doğrulanması gerekmektedir. Ancak bu doğrulama işlemi için kullanılabilecek kriptografik algoritma ve protokollerin, araç içerisindeki gerçek zamanlı haberleşme gereksiniminin göz önünde bulundurularak seçilmesi gerekir.

Anahtar Kelimeler: Denetleyici Alan Ağı, Mesaj Doğrulama Protokolü, Mesaj Doğrulama Kodu, Ağ Güvenliği, Tekrarlama Saldırısı

To My Family

ACKNOWLEDGMENTS

I would like to thank my supervisor, Assoc. Prof. Dr. Ali Doğanaksoy and my co-supervisor Assoc. Prof. Dr. Fatih Sulak for their persistent support, invaluable guidance, encouragement and endless patience during my thesis.

I would also like to thank my examining committee members Assoc. Prof. Dr. Murat Cenk and Assoc. Prof. Dr. Oğuz Yayla for their time they spared for me.

Also a special thanks to my dear friends Onur, Nurettin, Serdar and Ebru who always encouraged me. I am grateful to my colleagues Alper and Batuhan for their insightful comments. I want to thank İbrahim Arslan for his endless support that motivates me at the every stage of the work.

Finally, I must express my very profound gratitude to my family for providing me an unfailing support and a continuous encouragement throughout my years of study and through the process of researching and writing this thesis.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ALGORITHMS

LIST OF ABBREVIATIONS

CAN	Controller Area Network
ECU	Electronic Control Unit
AES	Advanced Encryption Standard
ACK	Acknowledge
CRC	Cyclic Redundancy Check
MAC	Message Authentication Code

CHAPTER 1

INTRODUCTION

1.1 Motivation and Definition of the Problem

Today's modern vehicles include dozens of Electronic Control Units (ECUs) that perform many essential functions such as braking, shifting gears and steering. Besides, the functionalities including autonomous driving, Vehicle-to-Vehicle (V2V) and Vehicle-to-Infrastructure (V2I) communication are expected to be fulfilled by ECUs. Various protocols such as Local Interconnect Network (LIN) [5], Controller Area Network (CAN) [7] and FlexRay [6] have been defined and implemented for connecting ECUs in the vehicle. CAN is the most widely used in-vehicle communication protocol in the automotive industry.

When the CAN protocol was first developed, security requirements were not considered as it would be used in a closed environment. However, the connection of the in-vehicle network to external networks through various interfaces as illustrated in Figure 1.1 brings security needs. These external interfaces allow malicious adversaries to access the vehicle system. In this way, they can compromise the safety of the driver and passengers by sending messages that may adversely affect the operation of the vehicle. Broadcast-based communication in the vehicle and the lack of an authentication mechanism in the CAN protocol cause the attacker to exploit the vulnerability by sending spoofed messages. Studies [10] have shown that vehicles can be easily targeted by malicious adversaries. The authors of [15] have demonstrated security vulnerabilities that can arise when the CAN is connected to the external world. Further studies [14] have shown that CAN is also vulnerable to DoS (Denial of Ser-

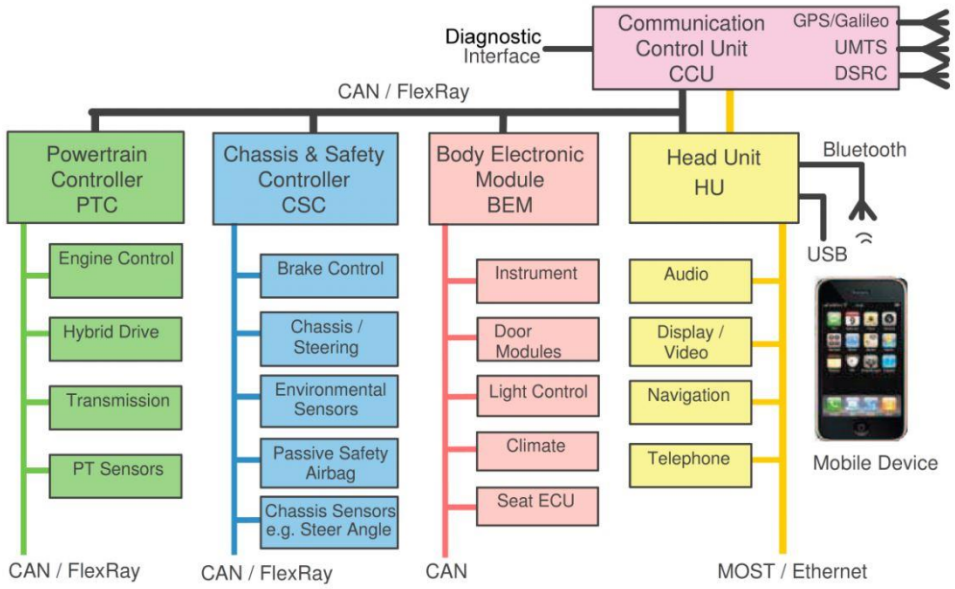

Figure 1.1: Typical in-vehicle network

vice) attacks.

1.2 Challenges in Security Solutions

Due to the nature of in-vehicle networks, there are many challenges that make it difficult to implement security solutions. First of all, ECUs in the in-vehicle network have limited processing power. Therefore, when proposing a security solution, it is necessary to consider the need for real-time communication, which is essential for in-vehicle networks. Since ECUs have limited storage space, the storage of keys and software required for cryptographic operations is another challenge. Moreover, it can be said that due to the low data rates of the CAN bus, the bus utilization should also be considered. Finally, the last challenge that must be overcome is that timestamps used to prevent replay attacks are not included in the in-vehicle network.

1.3 Related Work

This section discusses the solutions and different approaches that researchers have made to improve the security of the in-vehicle network. Lin *et al.* [11] proposed an authentication based security mechanism as a precaution against the masquerade

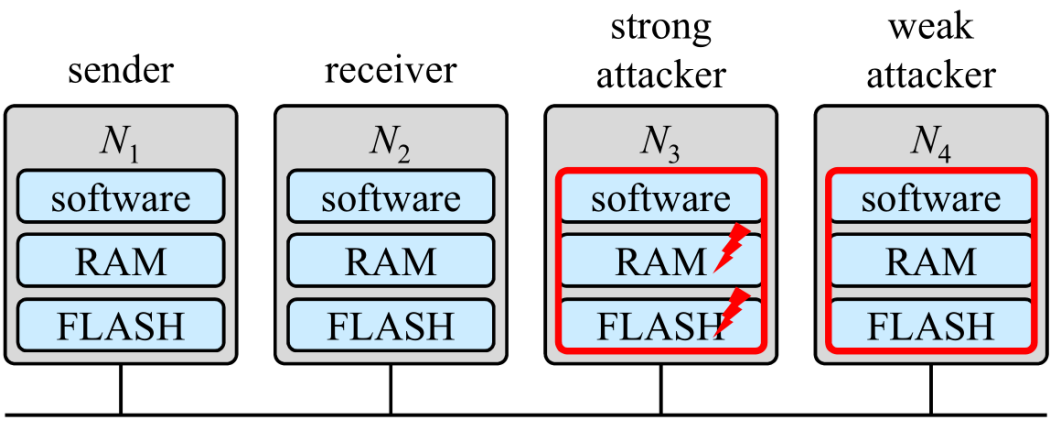

Figure 1.2: Attacker model [11]

and replay attack in the CAN. They defined an attacker model which is illustrated in Figure 1.2. While N_1 and N_2 are legitimate nodes, control of N_3 that stores critical data (e.g., shared secret keys) and N_4 that does not store any critical data is taken over by malicious software. N_3 and N_4 are called strong and weak attackers respectively. Fabrication is possible for N_3 since N_3 has access to shared secret keys and it can send a message to N_2, pretending to be N_1. On the other hand both strong and weak attackers (N_3 and N_4) are able to read a message that is sent by N_1 and then send the same message. N_2 will accept the message by considering it was sent by N_1. Therefore, both attackers can successfully perform a masquerade/replay attack unless the messages sent by nodes are verified without the message authentication protocol. Each node in the network stores ID table, the pair-wise symmetric key, and message counters in the security mechanism they proposed. ID table contains identifications of messages, node ID of the sender and the list of the node IDs of the receivers. Each node sends the counter information of related message and the authentication codes calculated for each receiver along with the message. The main problem with the approach that is mentioned above is the increase of the bus load if the message is received by a large number of nodes.

Siddiqui *et al.* [17] proposed a method for secure communication over CAN which requires hardware modifications for each ECU. The security solution is based on client-server architecture. During the manufacturing phase, each client authenticates with the server and is registered in a trusted environment. A physical layer cryptographic primitive which is called physical unclonable function (PUF) is embedded

into each node. During the registration process, PUF receives bit-string as a challenge and computes unique reproducible secret r as shown in the Figure 1.3. This secret and configuration parameters are used to generate a public-private key pair for Elliptic Curve Cryptography (ECC). The public key of each ECU is stored in a database of the server and each client node stores the public key of the server in their non-volatile memory. Each session starts with the turning on the ignition of the vehicle. At the beginning of the session, Shared Key Generation Block generates shared symmetric encryption key Sh_{ab} by using the private key of the node and public key of the server. Elliptic Curve Diffie-Hellmann (ECDH) algorithm is used for secret shared key exchange. After that, the encrypted public key of the node with the shared key is sent to the server. The server decrypts the message and checks whether the public key of the transmitter node is in the database of the server. If the server finds received public key in the database, it then considers the transmitter node as authorized for the rest of the session. The server shares public key information of all registered nodes with each client nodes by encrypting it using the related shared keys. Finally, each client node communicates with other nodes and generates a session key using its private key and other nodes' public key. The fact that private keys are only stored in volatile memory during the session and are reproduced at each session, provides a significant advantage in terms of security. Furthermore, this time-consuming authentication process in each session raises a question about whether the method is practical or not. The hardware improvements required at each node to implement this methodology, appear to be a major drawback in the automotive industry, where a cost-effective solution is a priority.

Kang $et\ al.$ [8] presented a lightweight source authentication protocol in their study. The attacker model that they consider is that adversaries perform replay attacks or masquerade attacks by listening to the bus and inserting CAN data frames. As shown in the Figure 1.4 their proposed protocol allows ECUs to use a one-way hash chain to prevent these types of attacks. Source authentication protocol consists of three phases: initialization, transmission and seed value sharing. At the initialization phase, the sender ECU constructs a one-way hash table using the first seed that is securely shared with other nodes. At the same time, the receiver ECUs compute the last hash value ($(n+1)$th value where n is the maximum number of hash values in the hash

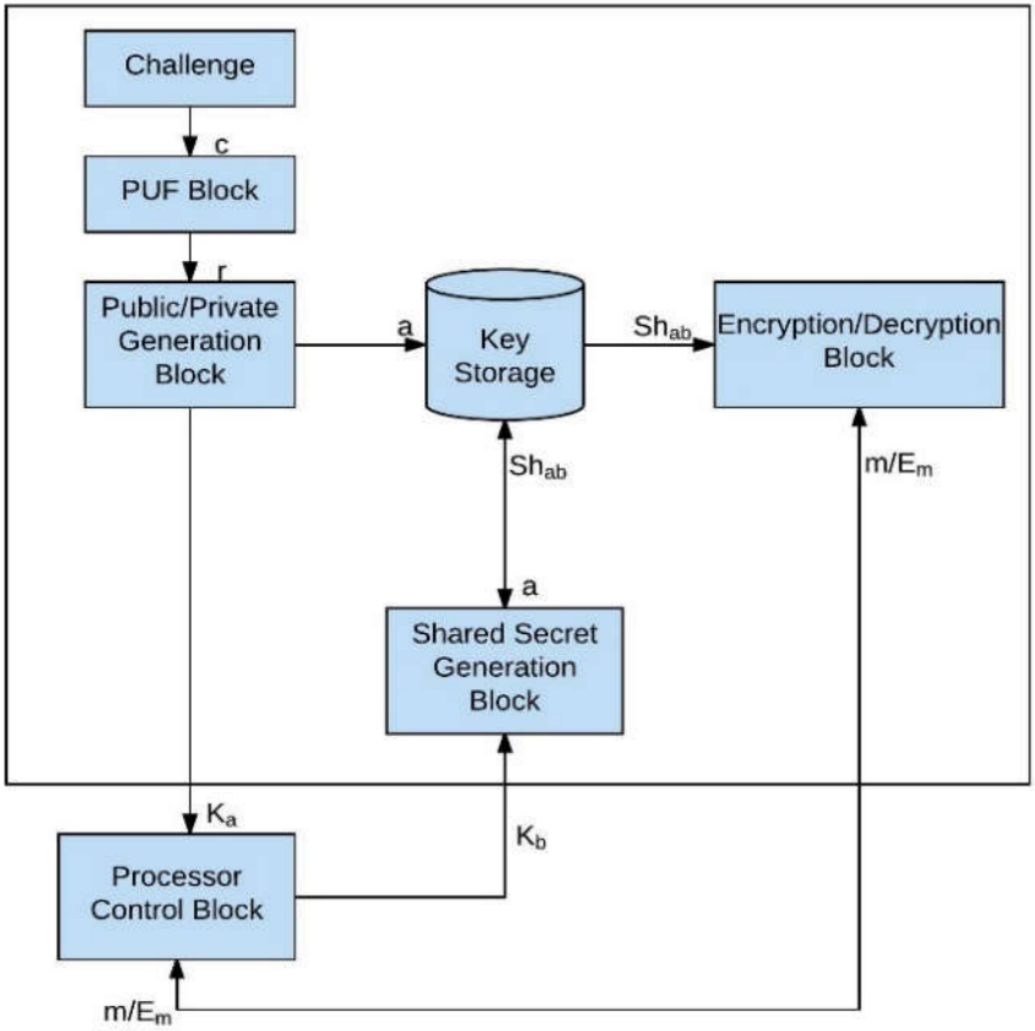

Figure 1.3: Secure block implemented at each client node [17]

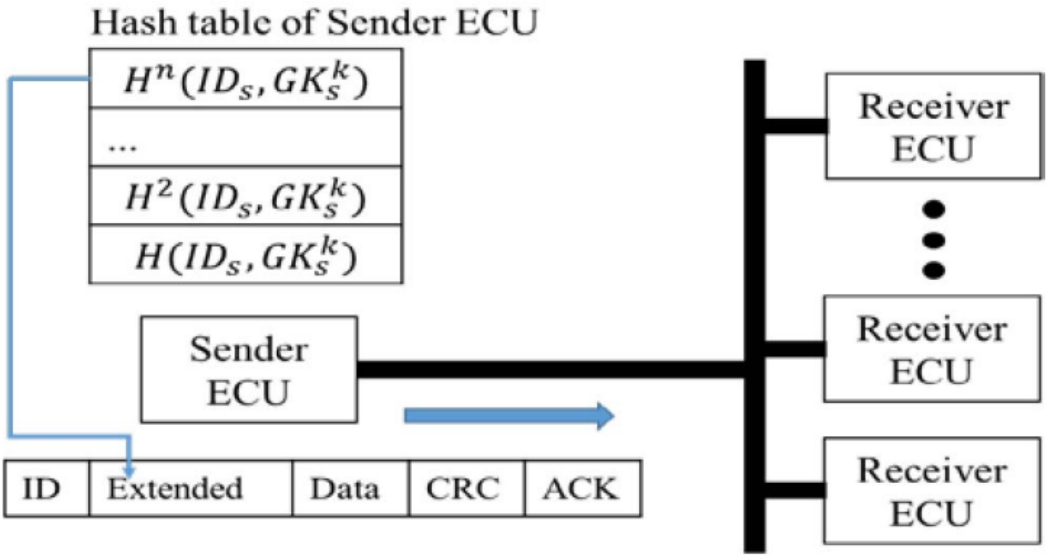

Figure 1.4: Source authentication protocol [8]

table) of the chain by using the same seed and the identifier of the sender ECU. At the transmission phase, the sender ECU sends the last hash value (nth for the first message) from its hash table along with the message. Receiver ECUs perform a hash operation to the received hash value to compare it with the stored one ($n + 1$th for the first message). In the seed value sharing phase, the next hash chain begins to be formed in parallel, taking into account the exhaustion of all hash values in the table. The proposed security mechanism has significant advantages over Keyed-Hashed Message Authentication Code (HMAC) based authentication protocol proposed by Samuel $et\ al.$ [20] in terms of authentication time, response time, and service delay according to experimental results. On the other hand, due to the fact that CAN controller filters are used to receive certain messages from the sender, a synchronization issue between the sender and the receiver nodes associated with the use of hash values seems likely to occur.

In another study, the authentication for verifying the identity of ECUs was performed using Message Authentication Code (MAC) based security framework that is presented by Wang $et\ al.$ in [19]. They separate ECUs by categorizing them as the low-trust group and the high-trust group. Since ECUs with external interfaces, such as OBD-II port and telematics are easily targeted by attackers as the attacker can exploit their exposed interfaces remotely, they become a part of the low-trust group. The remaining ECUs are included in the high-trust group. ECUs in a high trust group

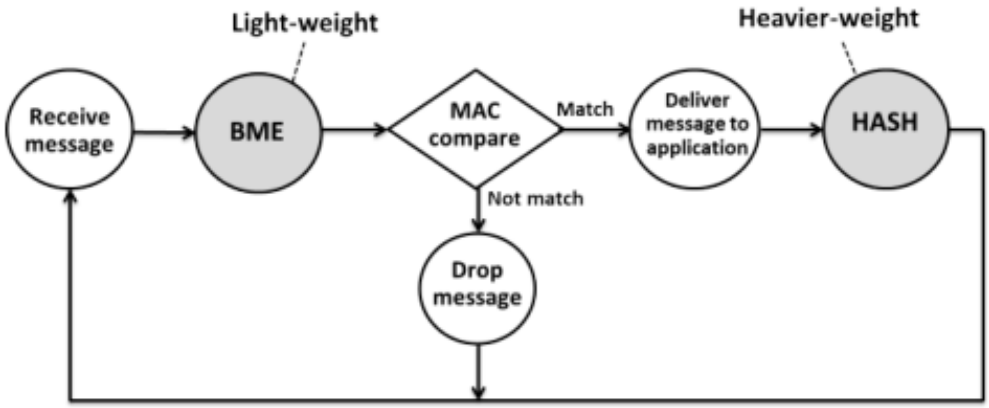

Figure 1.5: The message verification process at the receiver [19]

have the secret symmetric key required to authenticate messages unlike those in the low trust group. Nodes on the network send the data frame that contains the authentication message as well as the message that they originally want to send. Each authentication message contains 1-byte node ID, 2-byte message counter, 4-byte message authentication code and 1-byte authentication marker (0xFF). Since timestamp is inapplicable to vehicular systems, the message counter is used to uniquely identify a CAN message to resist replay attacks. The authentication marker serves to differentiate that the frame contains an authentication message. As shown in the Figure 1.5 to reduce the computational delay that occurs immediately before the data is sent, the digest that is independent of the data is pre-calculated. Using this digest and the Binding, Extraction and Mapping function called BME, which was described in detail in the relevant study, in authentication code generation and verification significantly improves computational performance. On the other hand, sending the authentication message for each data frame has a negative impact on the bus load of the CAN.

CHAPTER 2

CAN (CONTROLLER AREA NETWORK) PROTOCOL

CAN is a serial communication protocol that is widely used in in-vehicle communication. The CAN protocol was developed by Bosch in the early 1980s to meet the need for real-time communication between ECUs in vehicles. Although it was initially developed by the vehicle industry, today it is used in different areas such as industrial automation, management of medical equipment available in the operating room. In 1993 the CAN protocol was standardized as ISO 11898-1. A CAN bus with four nodes is demonstrated in Figure 2.1. In the CAN, the ECUs in the vehicle can be connected to each other by a single pair of wires instead of connecting each ECU with separate wires. This method provides an advantage in terms of weight, cost, and complexity.

CAN is a multi-master, broadcast-based communication system. Each node in the network broadcasts short messages that contain data obtained through sensors or other nodes. Because CAN is a carrier sense multiple access/collision avoidance (CSMA/CA) protocol, each node monitors whether the network is busy before sending messages to avoid a collision. The communication rate of the CAN can be up to

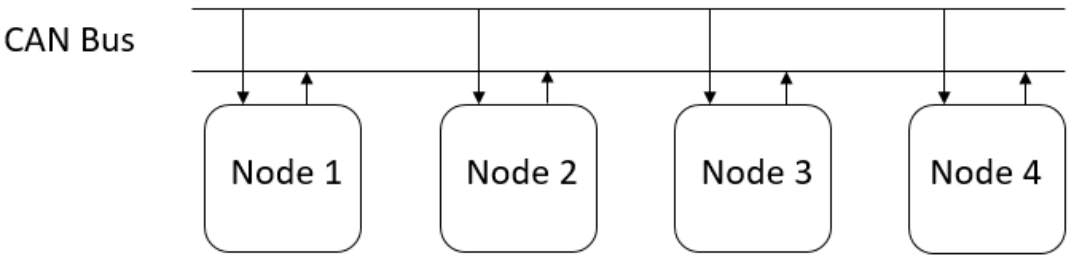

Figure 2.1: Four nodes connected through a CAN bus.

1 Mbps.

2.1 Message Frames

The original ISO standard CAN message have an 11-bits identifier. However, this standard was modified and the message identifier is extended to 29-bits which is called Extended CAN later on. Standard CAN messages are capable of having 2^{11}, or 2048 different identifiers, while Extended CAN messages can have 2^{29}, or 537 million different identifiers. Message identifiers include message priorities that allow arbitration. Data frames with 11-bits and 29-bits identifiers are shown in the Figure 2.2 and Figure 2.3, respectively.

CAN message frame starts with a dominant start-of-frame (SOF) bit that indicates the start of a message. It is followed by an 11-bits identifier which identifies the message and indicates the message's priority. The single remote transmission request (RTR) bit serves to distinguish whether the frame is a remote frame. A dominant RTR bit indicates a data frame while recessive RTR bit indicates a remote frame. Identifier extension (IDE) bit is dominant when a message with a standard CAN identifier is transmitted otherwise, IDE bit is recessive. The number of bytes that the data field contains is indicated in the 4-bits data length code (DLC). Up to 8 bytes of data in the data field can be transmitted. A 15-bit cyclic redundancy checksum (CRC) field allows the receiver to check the integrity of the received message. Every receiver in the network sends an ACK bit at the end of the message if they receive the message correctly. If ACK bit remains recessive, the sending node repeats the message. A 7-bit end-of-frame (EOF) field signifies the end of the CAN message. The interframe space (IFS) field provides the time required by the CAN controller to move the received message into its buffer.

There are four different types of CAN frames, including the data frame, the error frame, the remote frame, and the overload frame. The data frame consists of the identifier, the data, the data length code, the cyclic redundancy check, and the acknowledgment bits. RTR bit is dominant in the data frame. If any node detects an error in a message, it transmits an error frame. As a result, all the other nodes in the

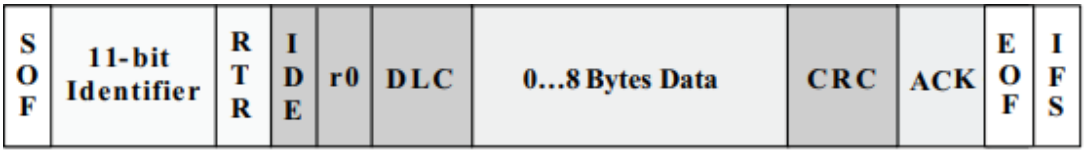

Figure 2.2: Standard CAN: 11-Bit Identifier [4]

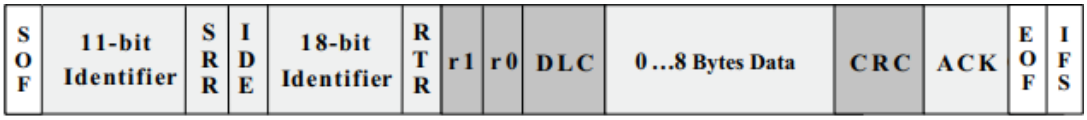

Figure 2.3: Extended CAN: 29-Bit Identifier [4]

network transmit error frames as well. In this case, the node that sends the erroneous message retransmits the message. Nodes in the network use a remote frame to request data from another node. The remote frame is similar to the data frame except that there is no data in it and the RTR bit is recessive. The overload frame is transmitted when a node does not have enough time to process the received message. This ensures that the receiving node has sufficient time to process the message.

2.2 Arbitration

One of the important characteristics of the CAN bus is the opposition in terms of logic states between the bus, and the driver input and the receiver output. The recessive level is associated with a logic one while the dominant level is associated with a logic zero on the CAN bus. As shown in Figure 2.4, the CAN controller sends the '010' bitstream and complementary of this bitstream takes place on the CAN-H line. CAN-L line's logic state is inverse of CAN-H line's logic state on the CAN bus.

Since all nodes on the network share the same physical communication bus, a collision may occur when two or more nodes attempt to access the bus simultaneously. This situation may cause undesirable effects, such as message destruction or corruption. Therefore CAN uses a bit-wise arbitration mechanism to avoid data collisions. The message identifier determines which node gains access to the bus during the arbitration phase. The message with the highest priority has a message identifier that consists entirely of zeros. During the arbitration phase, each transmitting node transmits its identifier and compares it with the level monitored on the bus. If these levels are equal, the unit continues to transmit. If the unit detects a dominant level on the

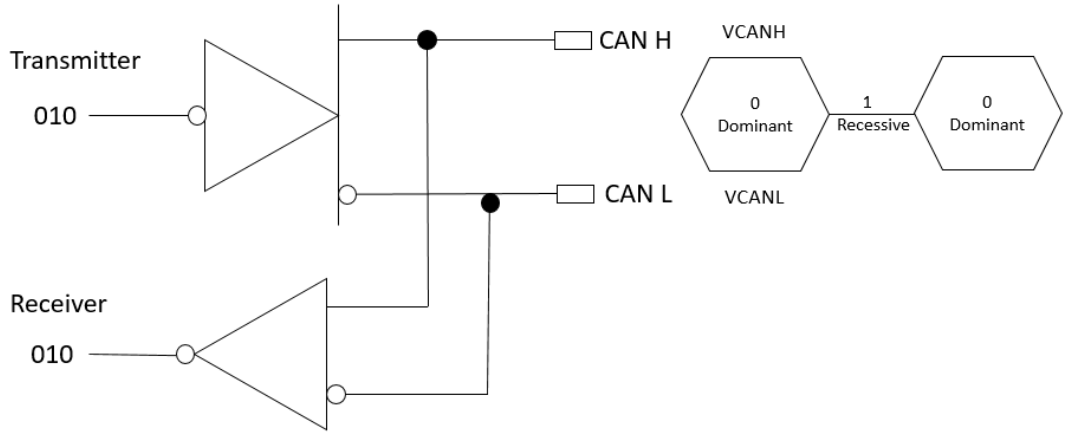

Figure 2.4: The inverted logic of a CAN bus

bus, while it was trying to transmit a recessive level, then it stops transmitting and switches to the listening mode.

Figure 2.5 shows an example where three nodes node A, node B and node C attempt to access the bus simultaneously. Message identifiers of each node are given below.

$$\mathrm{Id_A} = 10110100110 \;\; \mathrm{Id_B} = 10101101001 \;\; \mathrm{Id_C} = 10100100110$$

After all three nodes send the SOF bit, they start transmitting their message identifiers. When node A transmits the fourth bit of its message identifier as recessive bit and detects that bus state is dominant because of other node's message identifier, node A stops transmitting and starts listening. Node B also detects that its message identifier's recessive eighth bit is overwritten by node C and halts transmission. Therefore node C wins arbitration and continues sending the remaining part of the message. Nodes that lost arbitration starts a new arbitration as long as the bus is free to access again. Thus, non-destructive bus arbitration is provided by CAN.

2.3 Error Checking

The CAN protocol uses multiple error detection mechanisms to detect errors on the bus. The first of these mechanisms which is called bit monitoring is used by transmitter node by monitoring the signal on the bus. If there is any inconsistency between

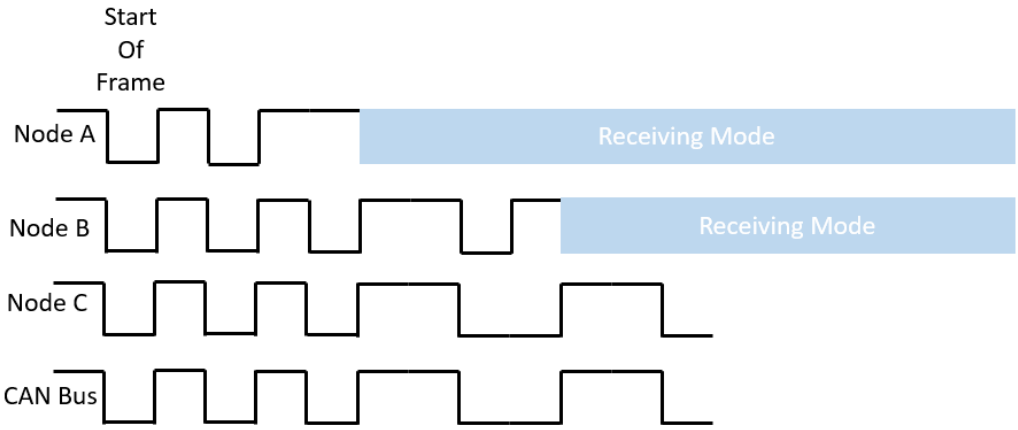

Figure 2.5: CAN bus arbitration example

the transmitted signal and the observed signal on the bus, the transmitter node sends an error frame. Thus the sender node is aware of an error as quickly as possible and alerts other nodes on the network. This operation is not performed during the arbitration phase. On the other hand, the data in the CRC field of the message is verified by the CRC which is calculated by each node in the network. A receiver that receives a valid message correctly, notifies a transmitter node by overwriting the ACK bit of a message and making it dominant. When the transmitter does not detect a dominant level on the bus during ACK transmission, it will notice that an error occurred during the transmission of the message. The sender can not send more than five same bits in a row somewhere between SOF and the end of CRC. After sending five identical bits, the transmitter sends a bit with the opposite value which is called the stuffed bit. Stuffed bits are ignored by the receiver nodes. Bit stuffing is also used for error detection. SOF, EOF, ACK delimiter, and the CRC delimiter bits are checked by receivers whether the values of these fields are consistent with the CAN specification.

CHAPTER 3

PROPOSED SECURITY PROTOCOL

In order to implement our proposed security solution, some prerequisites must first be met. Each ECU in the network stores a key encryption key and initial authentication key in their protected area. In the manufacturing phase, these keys are loaded through a secure channel. ECUs have a message ID table that contains the identification of received and transmitted messages. Since ECUs use message filtering functionality, the received messages that are not included in this table are ignored. The session counter and the counters of messages sent and received are kept in the volatile memory of each ECU to deal with the replay attacks. Because of their limited sizes, when transmitted and received message counters reach their maximum limits, they overflow. In order to avoid this situation, a session key is generated to be stored in a volatile memory of ECUs at the beginning of each session which is defined as turning on the vehicle's ignition. Each case the session key is generated, ECUs reset their received and transmitted message counters. The counter overflow which may occur due to the long duration of the session is prevented by updating the session key before the overflow occurs.

The notation list regarding our study is given in 3.1. Our proposed security mechanism consists of several phases such as initialization of ECUs, distribution of session keys, authentication of CAN messages and periodic update of session keys.

Table 3.1: Notation for proposed security mechanism

Notation	Description
ECU_k	k_{th} ECU in the network
ECU_m	Master ECU that is responsible for distribution of session key
SID_k	Source identifier of k_{th} ECU
MID_i	Message identifier of i_{th} message
K_{Enc}	Key encryption key
K_{Aut}	Initial authentication key
K_{Ses}	Session key
TMC_{MID_i}	Transmitted message counter of i_{th} message
RMC_{SID_k,MID_i}	Received message counter of k_{th} ECU and i_{th} message
SC	Session counter

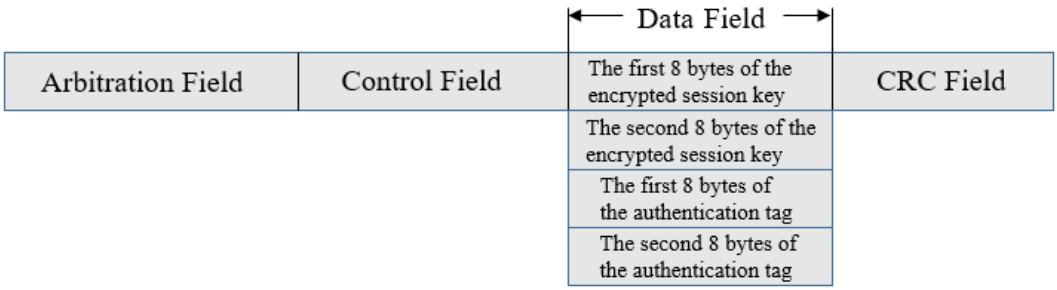

Figure 3.1: Session key message consisting of four data frames

3.1 Generation and Distribution of Session Keys

One of the ECUs in the network which is called master ECU is responsible for generating the session key and sharing it with other ECUs at the beginning of each session. This sharing process is easily completed by taking advantage of the CAN's broadcast-based communication by sending a total of four data frames, two of which contain the encrypted session key and the other two containing the authentication tag for the encrypted session key as shown in Figure 3.1. In this way, the session key is delivered to all ECUs as quickly as possible. As described in Algorithm 1, generation, and distribution of session key take place in a few steps. ECU_m;

i Generates a random number.

ii Using key derivation function obtains the session key.

iii Encrypts the session key with the key encryption key.

iv Calculates MAC of ciphertext by using the session counter and the initial authentication key.

v When the message is successfully transmitted, starts using a new session key and resets all transmitted and received message counters.

Algorithm 2 describes how ECUs in the network verify received session key message. ECUs except ECU_m;

i Calculate MAC of ciphertext obtained from received message by using its session counter and initial authentication key.

ii Compare received MAC with the calculated one.

iii If the verification is successful, start using the session key obtained by decrypting the ciphertext and reset all transmitted and received message counters.

Algorithm 1: Preparation of the session key message

Output: Session key message M

1 Generate a random number r.

2 With r as the input to the key derivation function, generate K_{Ses}.

3 Obtain ciphertext $C = E_{K_{Enc}}(K_{Ses})$.

4 Calculate $MAC = H_{K_{Aut}}(C||SC)$

5 Send cipher text C and MAC

6 Use the K_{Ses} for future communications.

7 Reset all TMC_{MID_i} and RMC_{SID_k,MID_i}

The session counter is stored by all ECUs in order to measure the replay attack that can be done during the sharing of the session key. Because the value of the session counter is incremented per session, even if the adversary copies one of the session key message sent before, this session key message will not be valid for another session.

Renewal of the session key at each session ensures that the session key's usage time is limited even if it is obtained by an attacker. In addition, since the session key is stored in the volatile memory, the attacker has a limited time to obtain the key.

Algorithm 2: Verification of the session key message

Input: Session key message M

Calculate $MAC_2 = H_{K_{Aut}}(C||SC)$.

if MAC_2 *is equal to MAC from received* M **then**

    Obtain session key $K_{Ses} = D_{K_{Enc}}(C)$.

    Use the K_{Ses} for future communications

    Reset all TMC_{MID_i} and RMC_{SID_k,MID_i}

else

    Ignore the message M.

3.2 Authentication of CAN Messages

After the session key is distributed, all ECUs within the network begin to send their messages together with a MAC. The study on how this MAC can be calculated most efficiently on the platform where we make our own implementation is described in the next chapter. But here it is worth noting that encryption and hash-based algorithms are used in order to calculate the MAC. Algorithm 3 describes what actions are performed on the sender's side before and after sending a message. The sender ECU;

i Finds transmitted message counter of the related message by using its identifier.

ii Concatenates message and its transmitted message counter. Also concatenates session key if it uses a hash-based algorithm.

iii Encrypts the result with session key or generate the digest of the result to obtain MAC depending on which algorithm it uses.

iv Sends a message and it's MAC.

v Monitors the CAN bus to determine whether the message was sent successfully.

vi Increments transmitted message counter if the transmission is successful

Algorithm 4 describes what actions are performed on the receiver's side to verify the received message. The receiver ECU;

Algorithm 3: Transmission of a message with an authentication tag

Input: Message m with identifier MID_i

1 Calculate $MAC = E_{K_{Ses}}(m||TMC_{MID_i})$ or
$MAC = H(m||TMC_{MID_i}||K_{Ses})$

2 Send the data frame with the message m and the MAC

3 **if** *Transmission is successful* **then**

4 Increment TMC_{MID_i}

5 **else**

6 Resend the data frame

i Finds received message counter of the related message by using it's identifier and source address.

ii Concatenates message and it's received message counter. Also concatenates session key if it uses a hash-based algorithm.

iii Encrypts the result with session key or generate a digest of the result to obtain MAC depending on which algorithm it uses.

iv Compares received MAC with the calculated one.

v If they are equal, increments related received message counter.

Algorithm 4: Verification of the received message

Input: Message m with message identifier MID_i and source identifier SID_k and MAC of message m

1 Calculate $MAC_2 = E_{K_{Ses}}(m||RMC_{SID_k,MID_i})$ or
$MAC_2 = H(m||RMC_{SID_k,MID_i}||K_{Ses})$

2 Compare the received MAC with the calculated MAC_2

3 **if** MAC *is equal to* MAC_2 **then**

4 Increment RMC_{SID_k,MID_i}

5 **else**

6 Authentication error

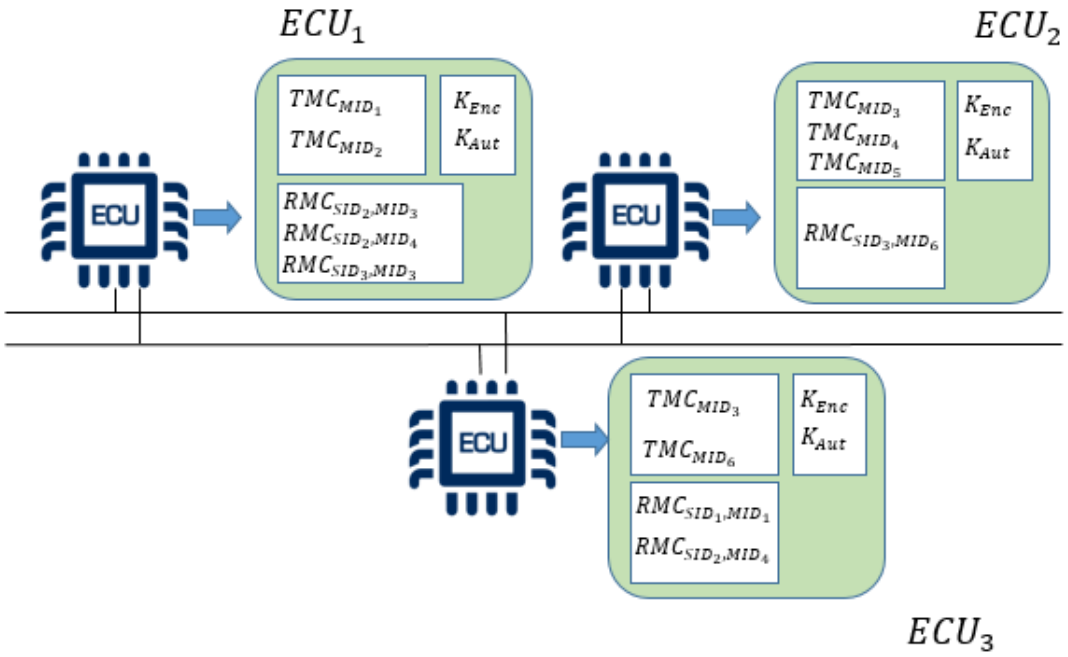

Figure 3.2: The stored data at the end of the initialization phase

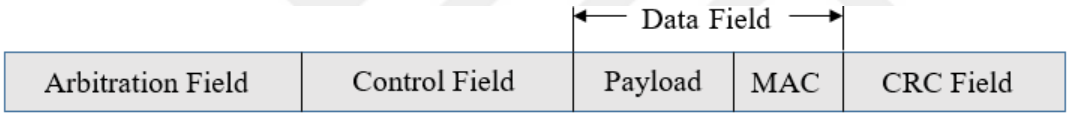

Figure 3.3: Data frame structure of the proposed protocol

Nodes do not receive every message shared on the network because they filter incoming messages. Therefore, when each node uses a common counter, there will be a synchronization problem between the nodes. The same type of messages within the network can be sent by multiple sources. Besides, a node may be receiving multiple types of messages from another node. For all these reasons, each node stores a transmitted message counter for each type of message it sends, and a received message counter for each type of message received from different sources, as shown in Figure 3.2.

The data frame used in the proposed security mechanism is as shown in Figure 3.3. The MAC calculated for each sent message is located within the data field. Figure 3.4 demonstrates a secure sharing of the message with the MID_1 identifier between ECU_1 and ECU_2.

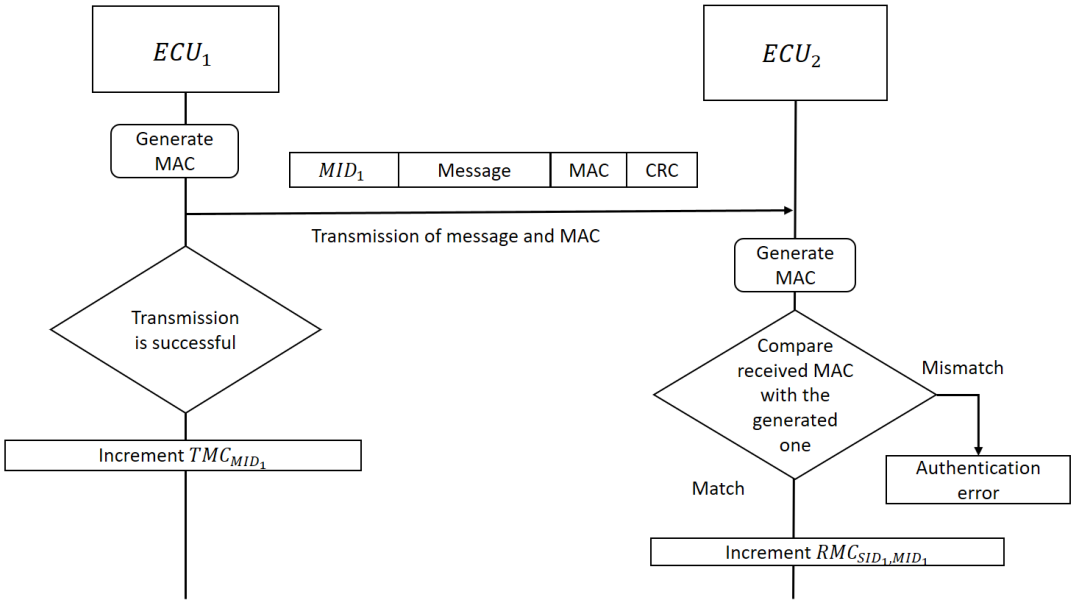

Figure 3.4: Authentication of a CAN data frame

3.3 Update of Session Keys

When any node in the network notices any of the message counters is approaching overflow, it requests the master node to generate a new session key. After the master node increases the value of the session counter by one, it generates a new session key, encrypts it, calculates the MAC required for the message, and broadcasts the new session key message. The nodes receiving the message, verify the authentication code of the message, then decrypt the corresponding part of the incoming message and start using the session key they obtained as shown in Figure 3.5.

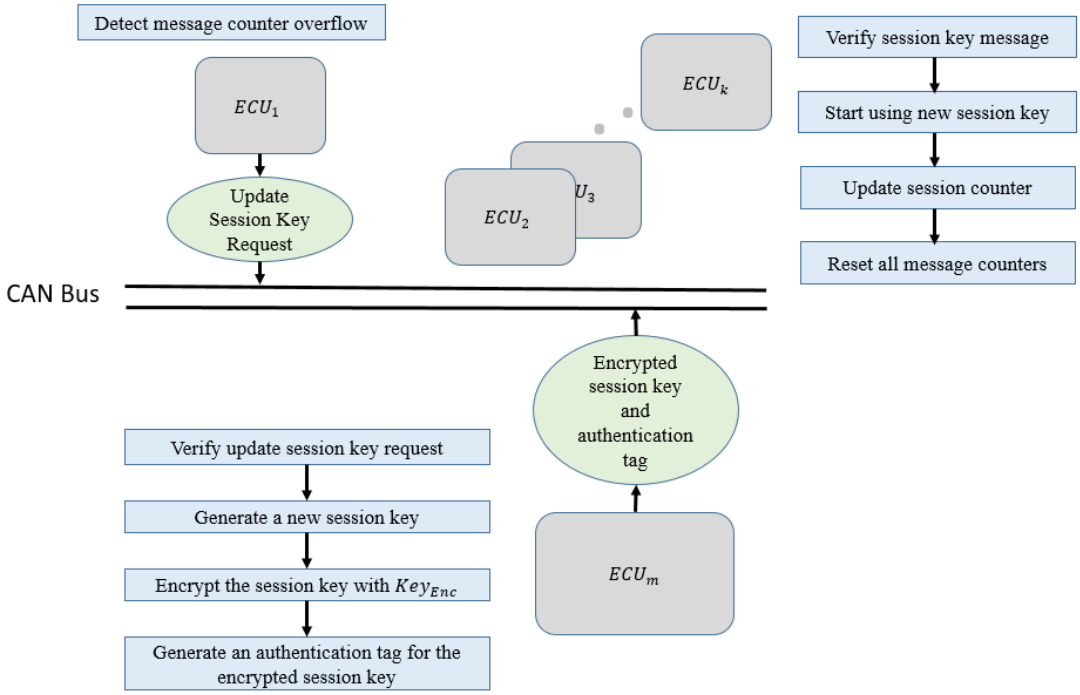

Figure 3.5: Session key update procedure

CHAPTER 4

IMPLEMENTATION AND PERFORMANCE ANALYSIS

In this chapter, we examined how the protocol we proposed is implemented and how the implementation we have done has the impact on the network along with the experimental results. When performing performance analysis, bus utilization and message latency on the bus were taken into consideration.

The software was developed on the 32-bit Arm Cortex-M4 based STMicroelectronics STM32F423ZH [18] microcontroller to obtain ECUs operating in accordance with the security protocol. To ensure communication between ECUs, bxCAN CAN controller which is included in the microcontroller was used. The Microchip CAN Bus Analyzer [13] was used to simulate the CAN bus environment and monitor the messaging traffic on the bus.

Different experiments were conducted by changing the number of ECUs in the network and the number of messages they send and receive. In addition, the performance of cryptographic algorithms used to obtain MAC was examined in two different categories such as hash-based ones and block cipher based ones.

In the protocol we propose, the payload length will be 6 bytes instead of 8 bytes since each data frame contains a MAC with a length of 2 bytes. The MAC length is limited due to the concern that the utilization of the CAN bus would increase considerably.

Performance evaluation was performed using block ciphers with a block size of 16 bytes such as AES [2], RC6 [16], Serpent [1], Twofish [9], Camellia [12], and Seed [3]. In this way, obtaining MAC by encrypting only one block has shortened the processing time. The 6-bytes message concatenated with the 4-bytes transmitted mes-

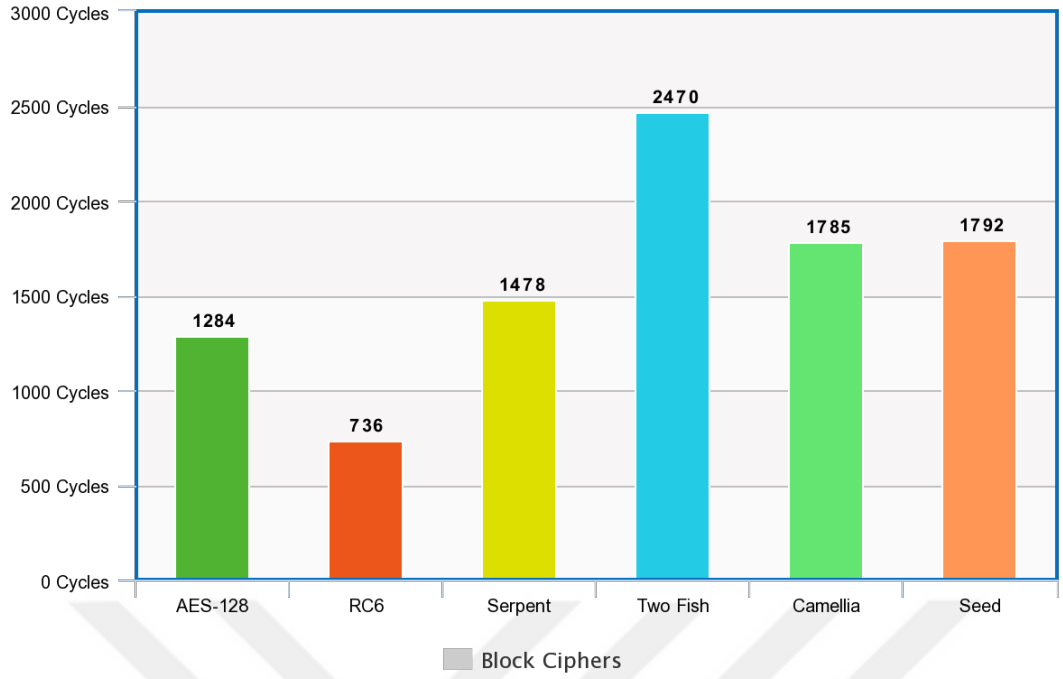

Figure 4.1: Authentication speed in clock cycles using different block ciphers

sage counter and then the 6-byte padding was applied to prepare the block to be encrypted. Figure 4.1 shows how much time it takes to prepare the block and create the MAC as a result of encryption. Besides, the speed of the message verification process at different operating frequencies of the processor is shown in the Figure 4.2.

When the MAC was created by using hash algorithms, a 6 byte message, a 4 byte transmitted message counter and a 16 byte session key were given as input to the hash function. The 2 bytes of the output of the hash function was used as MAC. Since the desired MAC size is 2 bytes, while hash functions produce larger output, in addition to speed measurements with specific hash algorithms, another speed measurement was made by determining the parameters of Keccak algorithm in accordance with the selected 32-bit CPU and the size of the desired output as shown in Figure 4.3. Figure 4.4 shows the message verification speeds of the processor with different operating frequencies when obtaining MAC using hash algorithms.

Table 4.1: Speed evaluation conditions

CAN Bus Speed	1 Megabits per second
Algorithm	AES-128
Sent Message Time Interval	10 ms.

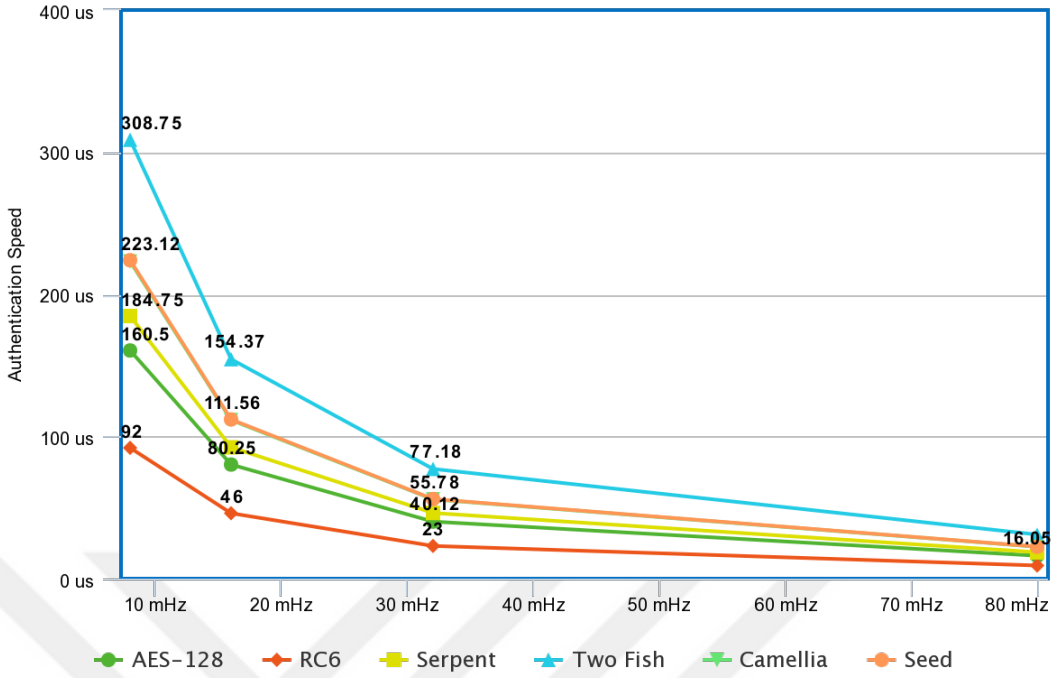

Figure 4.2: Authentication speed in μs using different block ciphers for different operating clock frequencies of processor

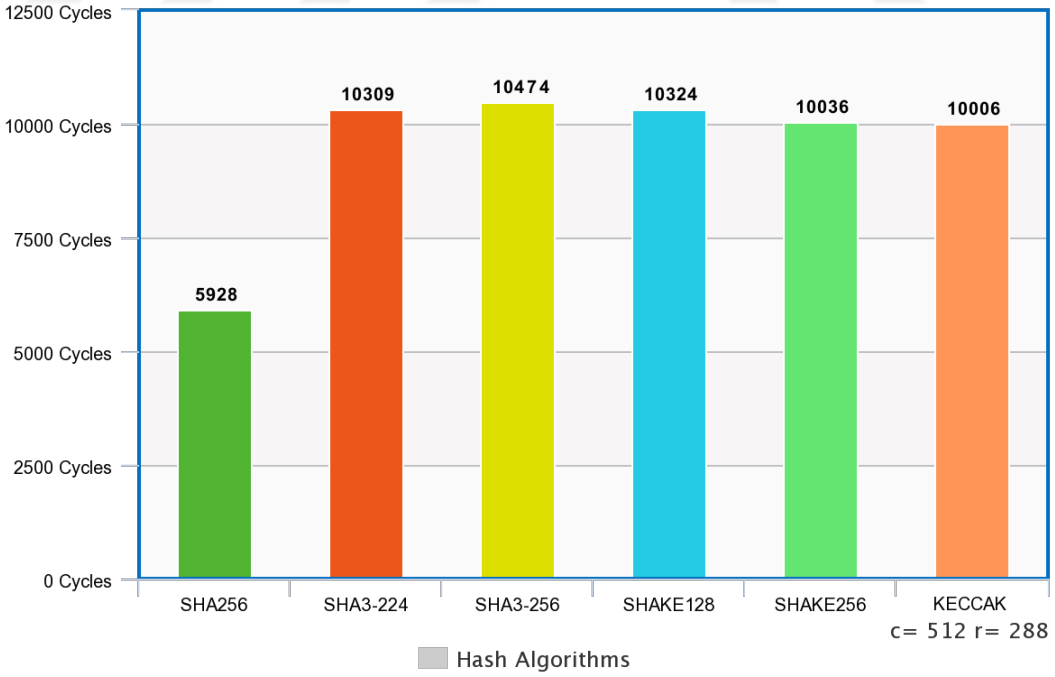

Figure 4.3: Authentication speed in clock cycles using different hash algorithms

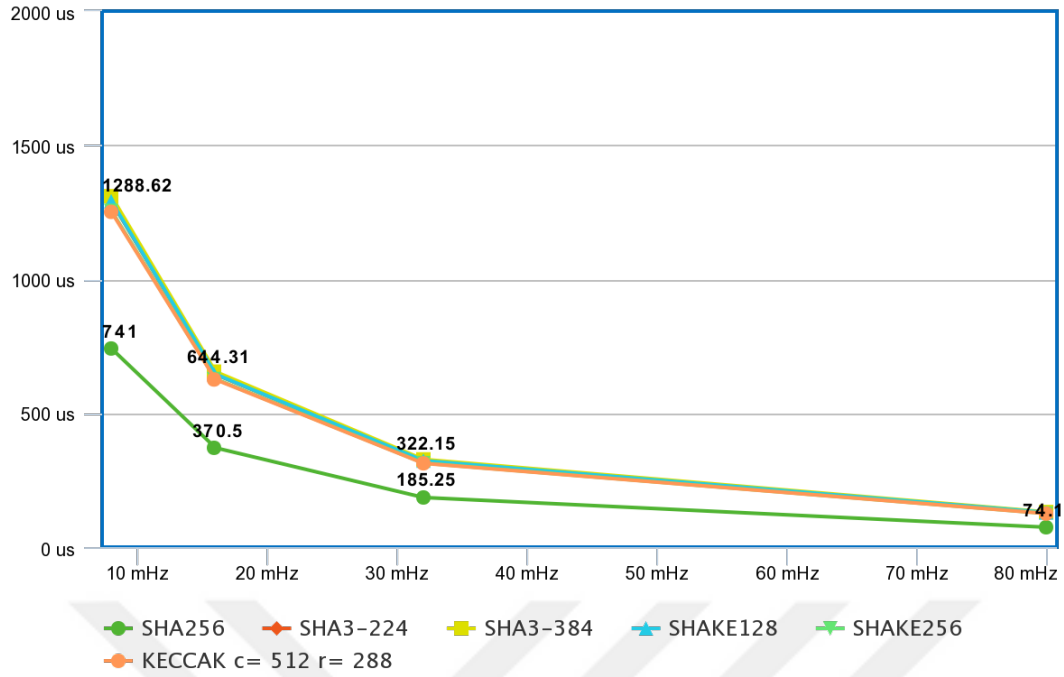

Figure 4.4: Authentication speed in μs using different hash algorithms for different operating clock frequencies of processor

Experiments performed in the rest of the thesis are based on the values in Table 4.1. In the Figure 4.5, the operating frequency of the receiving ECU's processor is 80 Mhz. Channel 4 of the logic analyzer is connected to the CAN bus, while channel number 3 is connected to a pin where the receiver ECU changes its state in certain cases. Channel 4 shows a CAN data frame sent in accordance with the proposed CAN protocol. On the other hand, channel 3 shows that the receiving ECU changed its pin state as soon as the message arrives and completes the verification process. The time for the sent message to reach the receiving node is 143 μs, while the verification time of the message is 16.8 μs. These times are indicated by timing marker pairs A1-A2 and B1-B2, respectively. Similarly, the results obtained by changing the operating frequencies of the receiving node are shown in Figure 4.6 and Figure 4.7. When the CPU clock rate is 60 Mhz, in another study [20], the authentication time is approximately 150 μs, whereas in the proposed study, this time is 22.4 μs. Furthermore, even if the CPU clock rate is 16 MHz, the authentication time is 83 μs, which is a significant improvement compared to the proposed method in the [20].

To measure round trip time, we connected another ECU with the same hardware

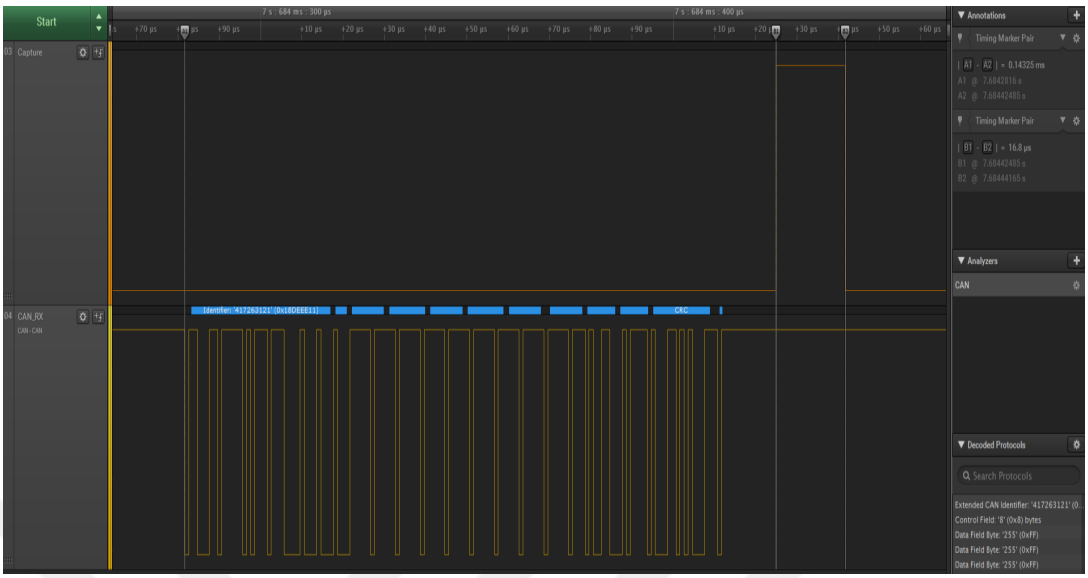

Figure 4.5: Time to receive and verify the message sent when the CPU is running at 80 Mhz

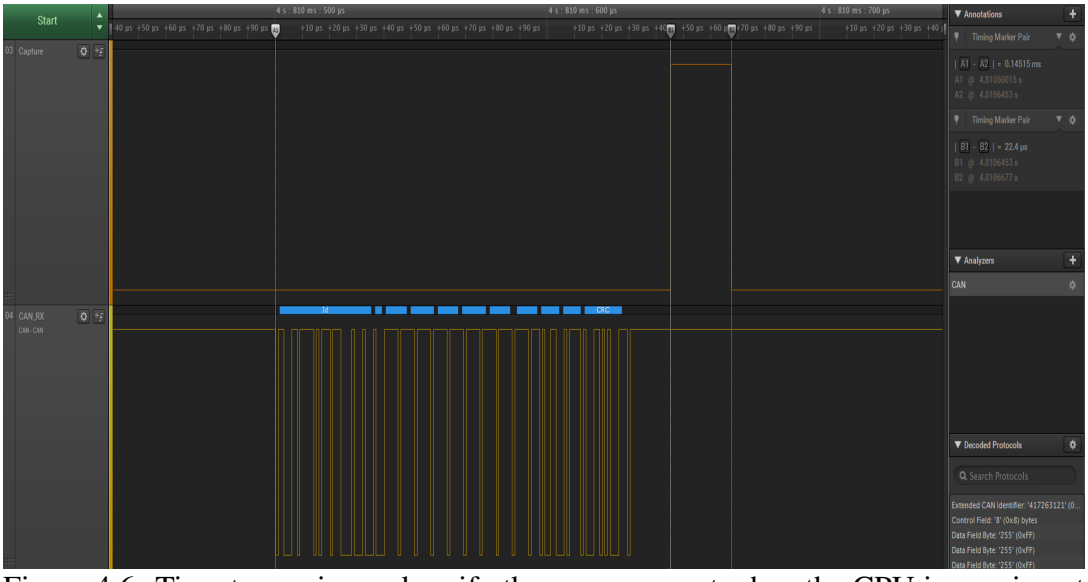

Figure 4.6: Time to receive and verify the message sent when the CPU is running at 60 Mhz

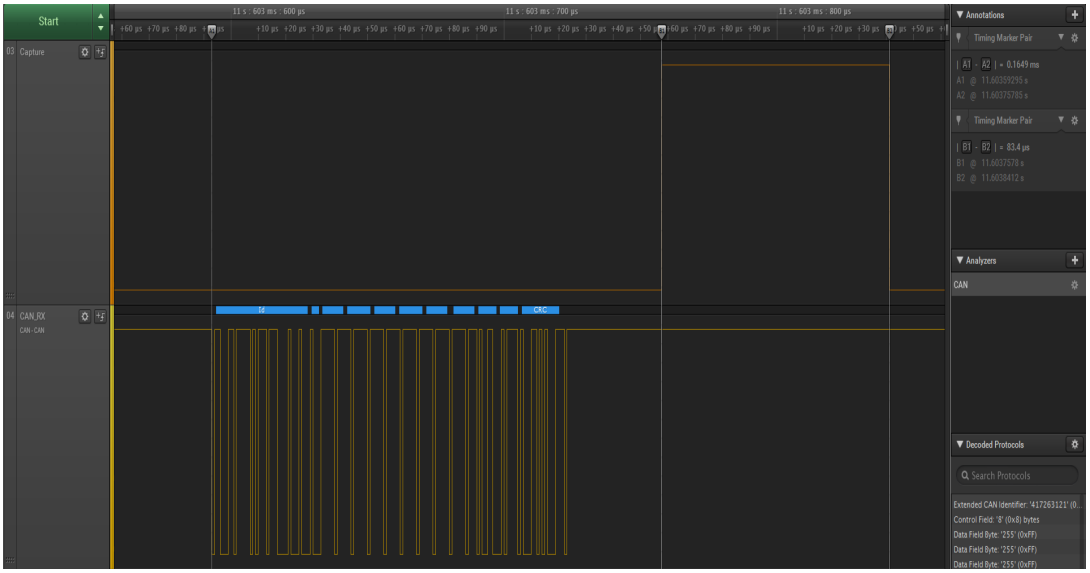

Figure 4.7: Time to receive and verify the message sent when the CPU is running at 16 Mhz

characteristics as the receiver ECU to the CAN bus. A large number of threads which is called virtual sender ECUs were executed on the ECU we connected. Each of these threads has the role of sending messages in 10 ms intervals. When the receiving ECU receives a message, it verifies this message and sends a response message with a MAC. In this way, the receiving ECU performed cryptographic calculations both when verifying the incoming message and sending the response message. Round trip time was calculated by adding up the time it takes to send a message and the time it takes to receive a response message. In Figure 4.8, the timing marker pair A1-A2 shows the total time it takes for a message to be sent and a response to be received, and the timing marker pair B1-B2 shows the time it takes to verify the incoming message and generate a verifiable response message. The number of threads was increased to see if the receiving ECU missed any messages. While receiving ECU was operating at a frequency of 16 MHz, there was no message loss when 25 virtual ECUs were sending messages at intervals of 10 milliseconds. However, it should be noted that the CAN controller of the receiving ECU has 2 receiver buffer. A message loss occurs if the number of senders exceeds 2 or 3 in 40 and 60 Mhz clock rate in the study of Samuel *et al.* [20], respectively according to study in [8]. Figure 4.9 shows that 25 different threads acting as 25 different ECUs send messages in 10 milliseconds intervals.

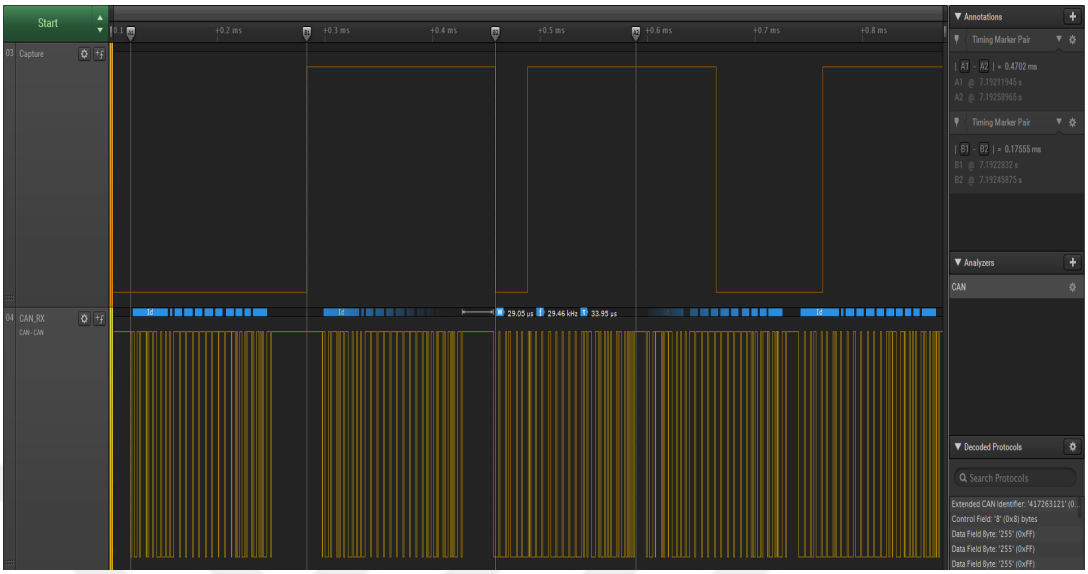

Figure 4.8: The round trip time when the CPU is running at 16 Mhz

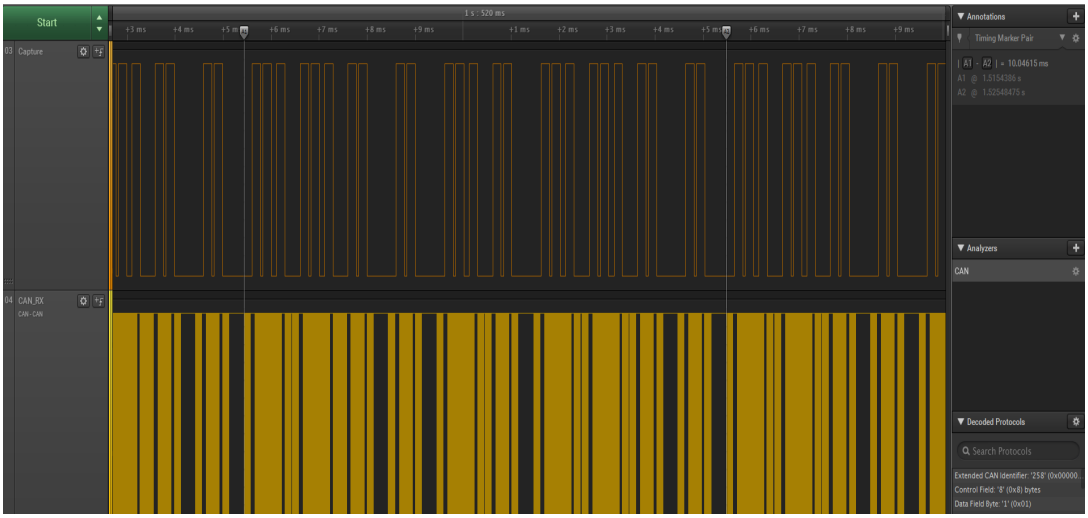

Figure 4.9: The CAN bus load when 25 senders send messages at 10 ms intervals

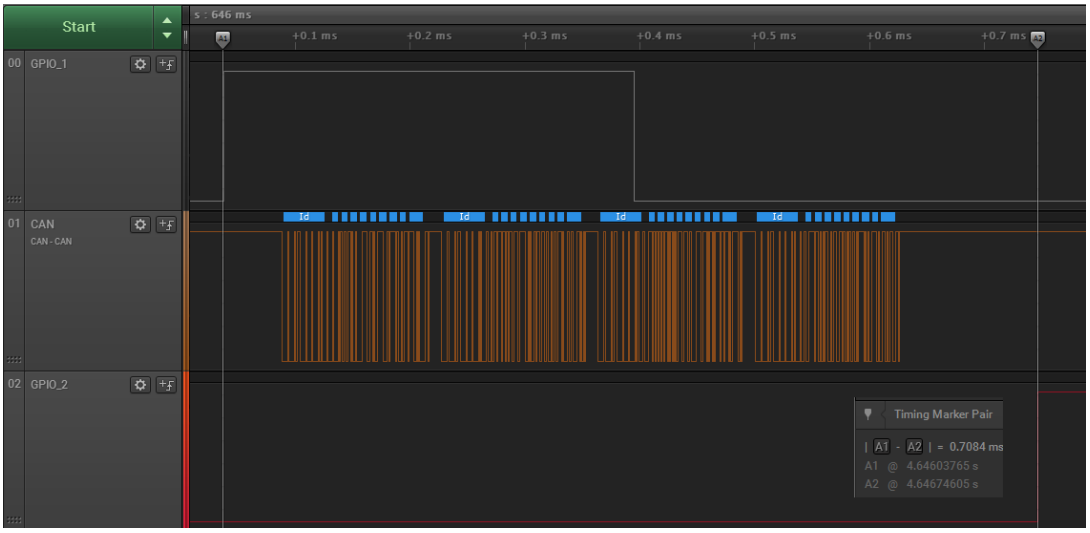

Figure 4.10: Session key update time when the CPU clock rate is 16 Mhz

We also measured the session key update time. Two ECUs are connected, one responsible for generating and distributing the session key, the other responsible for verifying the incoming key update message. Changing the number of receiver ECUs had no effect on the duration of the key update process, as all receiver ECUs would verify the incoming key update message at the same time. The process of updating the session key involves generating the session key, distributing the encrypted session key and authentication tag within four data frames and verification of the received data frames by receiver ECUs. Therefore, the session key update time is the sum of the duration of these processes. As shown in the Figure 4.10, the session key generation process starts when the state of the channel referred to as $GPIO1$ is logical high. Then the generated key is encrypted, the authentication tag is calculated and the four prepared data frames are written into the transmit buffers of the CAN controller. The GPIO state is set as logical low, as can be seen in the channel referred to as $GPIO1$. Receiver ECU changes the state of the line expressed in $GPIO2$ immediately after receiving and verifying the incoming message. The duration of the session key update process is 708 microseconds while the CPU clock rate is 16 Mhz, as shown by the A1-A2 timing marker pair.

CHAPTER 5

CONCLUSION

In this thesis, a secure communication protocol was developed to prevent masquerade and replay attacks for the security of the vehicle network, which has become increasingly important with the opening of vehicles to the outside world. We proposed a software solution that is easy to implement and does not require any changes to the CAN protocol. The protocol was implemented using different types of cryptographic algorithms on a platform which is used intensively in IoT (Internet of Things) devices. The performance of the proposed protocol was evaluated considering different criteria. The effect of the time required for the generation and verification of the message authentication code on the ECUs was minimized. The proposed protocol was demonstrated to be efficient in terms of authentication time, response time and round trip time.

REFERENCES

[1] R. Anderson, E. Biham, and L. Knudsen, Serpent: A proposal for the advanced encryption standard, 2000.

[2] J. Daemen and V. Rijmen, *Rijndael/AES*, pp. 520–524, Springer US, Boston, MA, 2005, ISBN 978-0-387-23483-0.

[3] S. J. H.J.Lee, J.H.Yoon, The seed cipher algorithm and its use with ipsec, 2015.

[4] T. Instruments, Introduction to the controller area network (can), 2016, http://www.ti.com/lit/an/sloa101b/sloa101b.pdf.

[5] International Organization for Standardization, *Road vehicles — Local Interconnect Network (LIN) - Part 1: General information and use case definition*, ISO/DIS 17987-1, 2006.

[6] International Organization for Standardization, *Road vehicles — Communication on FlexRay — Part 1: General information and use case definition*, ISO 10681-1, 2010.

[7] International Organization for Standardization, *Road vehicles — Controller area network (CAN) - Part 1: Data link layer and physical signalling*, ISO11898-1, 2015.

[8] K. Kang, Y. Baek, S. Lee, and S. H. Son, An attack-resilient source authentication protocol in controller area network, in *2017 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, pp. 109–118, 2017.

[9] B. Kelsey, D. Whiting, D. Wagner, C. Hall, and N. Ferguson, Twofish: A 128bit block cipher, 1998.

[10] 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 *2010 IEEE Symposium on Security and Privacy*, pp. 447–462, 2010, ISSN 1081-6011.

[11] C. Lin and A. Sangiovanni-Vincentelli, Cyber-security for the controller area network (can) communication protocol, in *2012 International Conference on Cyber Security*, pp. 1–7, 2012.

[12] S. M. M. Matsui, J. Nakajima, A description of the camellia encryption algorithm, 2004.

[13] Microchip, Can bus analyzer user's guide, 2011, http://ww1.microchip.com/downloads/en/DeviceDoc/51848B.pdf.

[14] P.-S. Murvay and B. Groza, Dos attacks on controller area networks by fault injections from the software layer, in *Proceedings of the 12th International Conference on Availability, Reliability and Security (ARES 2017)*, pp. 1–10, 2017.

[15] D. K. Nilsson, U. E. Larson, F. Picasso, and E. Jonsson, A first simulation of attacks in the automotive network communications protocol flexray, in E. Corchado, R. Zunino, P. Gastaldo, and Á. Herrero, editors, *Proceedings of the International Workshop on Computational Intelligence in Security for Information Systems CISIS'08*, pp. 84–91, Springer Berlin Heidelberg, 2009, ISBN 978-3-540-88181-0.

[16] R. L. Rivest, M. J. B. Robshaw, R. Sidney, and Y. L. Yin, The rc 6 block cipher, 1998.

[17] A. S. Siddiqui, Y. Gui, J. Plusquellic, and F. Saqib, Secure communication over canbus, in *2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS)*, pp. 1264–1267, 2017.

[18] STMicroelectronics, Stm32f423xh, 2017, www.st.com/en/microcontrollers-microprocessors/stm32f423zh.html.

[19] Q. Wang and S. Sawhney, Vecure: A practical security framework to protect the can bus of vehicles, in *2014 International Conference on the Internet of Things (IOT)*, pp. 13–18, 2014.

[20] S. Woo, H. J. Jo, and D. H. Lee, A practical wireless attack on the connected car and security protocol for in-vehicle can, IEEE Transactions on Intelligent Transportation Systems, 16(2), pp. 993–1006, 2015, ISSN 1558-0016.