Cairo University

METHODS OF SECURING IN-VEHICLE NETWORKS

by

Ahmed Hazem Gamal Yousef

A Thesis Submitted to the
Faculty of Engineering at Cairo University
in Partial Fulfillment of the
Requirements for the Degree of
MASTER OF SCIENCE
in
ELECTRONICS AND COMMUNICATIONS ENGINEERING

FACULTY OF ENGINEERING, CAIRO UNIVERSITY
GIZA, EGYPT
2013

METHODS OF SECURING IN-VEHICLE NETWORKS

by

Ahmed Hazem Gamal Yousef

A Thesis Submitted to the
Faculty of Engineering at Cairo University
in Partial Fulfillment of the
Requirements for the Degree of
MASTER OF SCIENCE
in
ELECTRONICS AND COMMUNICATIONS ENGINEERING

Under the Supervision of

Assoc. Prof. Dr. Hossam Aly Hassan Fahmy

Associate Professor
Electronics and Communications Department
Faculty of Engineering, Cairo University

FACULTY OF ENGINEERING, CAIRO UNIVERSITY
GIZA, EGYPT
2013

METHODS OF SECURING IN-VEHICLE NETWORKS

by

Ahmed Hazem Gamal Yousef

A Thesis Submitted to the
Faculty of Engineering at Cairo University
in Partial Fulfillment of the
Requirements for the Degree of
MASTER OF SCIENCE
in
ELECTRONICS AND COMMUNICATIONS ENGINEERING

Approved by the
Examining Committee

Assoc. Prof. Dr. Hossam Aly Hassan Fahmy, Thesis Main Advisor

Prof. Dr. Magdy Saeed El-Soudani, Member

Prof. Dr. Ayman Mohamed Mohamed Hassan Wahba, Member

FACULTY OF ENGINEERING, CAIRO UNIVERSITY
GIZA, EGYPT
2013

Engineer's Name:	Ahmed Hazem Gamal Yousef
Date of Birth:	03/07/1983
Nationality:	Egyptian
E-mail:	ahmedhazem@ieee.org
Phone:	0100 181 0215
Address:	42 El Sheikh Aly A.Razik St., Cairo.
Registration Date:	01/10/2007
Awarding Date:	.../.../....
Degree:	Master of Science
Department:	Electronics and Communications Engineering

Insert photo here

Supervisors:

Dr. Hossam A. H. Fahmy

Examiners:

Prof. Ayman M. Wahba (Professor at Faculty of Engineering - Ain Shams University)
Prof. Magdy S. El-Soudani
Dr. Hossam A. H. Fahmy

Title of Thesis:

Methods of Securing In-Vehicle Networks

Key Words:
Automotive; Security; Authentication; CAN; Protection

Summary:

The design of in-vehicle networks has been always concerned with reliability and safety rather than security. Experiments have demonstrated practical attacks on different systems inside vehicles. The thesis focuses on achieving protection of the CAN bus against adversaries. The proposed method is to introduce message source authentication protocol for the CAN bus. This is achieved by designing a new lightweight authentication protocol.

Acknowledgements

First of all, I have to thank God for guiding me to finishing this work. I would like to thank my parents for their support and encouragement.

I would like to thank my academic advisor Dr.Hossam for his guidance and support throughout the previous years; he taught me how to do research, how to think in an organized way and then how to document the work in a perfect way both technically and linguistically.

I would like to thank Karim El-Defrawy for the fruitful discussion that we had about security challenges in the automotive industry that opened the door for the main topic of the thesis.

I would like to thank Prof. Dr. Magdy El-Soudani for teaching me foundations of cryptography during my graduate course studies.

Table of Contents

List of Tables

List of Figures

Nomenclature

Table 1: List of Abbreviations

ECU	Electronic Control Unit
CAN	Controller Area Network
OBD	On Board Diagnostic
TPMS	Tire Pressure Monitoring System
RFID	Radio Frequency IDentification
HVAC	Heat Ventilation and Air Conditioning
HMI	Human Machine Interface
GPS	Global Positioning System
DoS	Denial of Service
AES	Andvanced Encryption Standard
TESLA	Timed Efficient Stream Loss-tolerate Authentication
MAC	Message Authentication Code
SHA	Secure Hash Algorithm
ADC	Analog to Digital Converter
HMAC	Keyed-Hash Message Authentication Code
NIST	National Institute of Standards and Technology
MOST	Media Oriented Systems Transport
LIN	Local Interconnect Network
ECRYPT	European Network of Excellence in Cryptology
RAM	Random Access Memory
RKE	Remote Keyless Entry
HSM	Hardware Security Module
WMA	Windows Media Audio
IRC	Internet Relay Chat
EVITA	E-safety Vehicle Intrusion proTected Applications
CPA	Correlation Power Analysis

Abstract

The evolution of electronics in the previous century has changed the nature of vehicles dramatically. Nowadays, the control of different systems within a vehicle is carried out using tens of electronic control units (ECUs). These ECUs are clustered into networks having gateways in-between. Several standards are used for the communication within these networks and between them. The most widely used standard is the Controller Area Network (CAN) bus standard. In general, the design of such 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. Practical experiments have demonstrated practical attacks on different systems inside vehicles. This included the ECUs controlling engine, brakes, lighting, climate control lighting and body controller. As a result, it is possible for an adversary to take-over the control of a vehicle and harm the passengers. This highlights a major risk for all cars; whether they have already been sold or they are still under development.

There are two types of security vulnerabilities that make these attacks possible. The first type is due to the inherent weaknesses of the used communication standards themselves. The second type is due to the deviation from security standards. The goal of this thesis is to investigate different methods by which the security of in-vehicle communication networks can be improved. The proposed methods consist of different levels of security enhancements ranging from prevention, protection and detection. The thesis focuses on achieving protection of the CAN bus against adversaries. The proposed method is to introduce message source authentication protocol for the CAN bus. This is achieved by designing a new lightweight authentication protocol. The protocol is implemented on an automotive grade microcontroller. The main aspect of the protocol is that it is simple and practical so that it can be adopted directly in the automotive industry. The protocol does not need any hardware modifications and thus can be deployed inside cars that have already been sold.

Chapter 1

Introduction

1.1 Cars as embedded systems

1.1.1 Evolution of cars

The 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) [1]. 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 reflashing can be made during regular service that is made in authorized workshops.

1.1.2 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 all car networks are linked together through gateways, 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 [2]. 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. That's why information security has not been introduced in the automotive industry except for specific systems like immobilizers.

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 OBD II port (used for diagnostics) 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 [3]...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 or within the network itself or both.

This situation is similar to the case of personal computers during the spread of internet usage. Before the internet, the spread of computer viruses was limited to using infected removable media (floppy discs by that time and compact discs later on). Computer users already had their own precautions to protect their computers from those threats. With the emergence of the internet, personal computer users found themselves susceptible to new types of threats that may spread on a global level. It took some time for security systems to become mature and provide and appropriate protection. Unfortunately, the automotive industry nowadays is facing the same problem again.

1.2 Experimental analysis of attacks

All the threats discussed above have always been treated as impractical until Koscher et al [2] 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. This section summarizes the work done in the two parts and the obtained results.

1.2.1 Direct access

Direct physical access to the car network was feasible through the OBD II port. Three test methods were used to realize the attacks:

- Isolating the ECU in the lab and connecting it to benches using harness
- Elevating the car on jacks (Stationary car)
- Doing a road test

Exploring CAN messages

There were many attack methodologies that were used to know details about the CAN network:

- **Packet sniffing**. They developed a tool to analyze packets that are being transmitted on the CAN bus. By observing regular messages that are being transmitted on the bus after requesting some commands, it was possible to discover how to control the radio, Instrument Panel Cluster, as well as many functions of the body controller.
- **Fuzzing**. By injecting random messages on the CAN bus and observing the result, it was possible to discover many of DeviceControl [1] functions. Actually, fuzzing itself can be an effective attack.
- **Reverse engineering**. For some ECUs, it was possible to dump the code using ReadMemory service through the CAN bus and then understand the structure of the code using a third party debugger.

[1]DeviceControl is a diagnostic service that is used to do some diagnostic tests during the service of the car. When it is used, it is possible to control devices in the car regardless of the normal behavior imposed by the current user request

Successful attacks

After knowing the structure of CAN messages that are exchanged within the car network, it was possible to realize the attacks on some car systems.

Radio . It was easy to gain full control of the radio and its display disabling the user interface. As a result, it was possible to produce arbitrary sounds and alarms of different volume levels.

Instrument Panel Cluster . It was possible to fully control the display of the instrument panel cluster, displaying arbitrary messages, changing the speedometer reading, fuel level and back-lighting.

Body Controller . Using both reverse engineering and fuzzing, it was possible to know how to control most of the body controller functionalities. It was possible to lock/unlock the doors, enable/disable power windows, adjust internal/external lighting levels, open the trunk, turn on/off the horn, enable/disable wipers and continuously shoot the windshield fluid, disable/enable the key lock solenoid in order to either lock the key in the car or make it possible to take the key off the car while the engine is still running.

Engine . Using fuzzing for DeviceControl, it was possible to control the engine in many ways. For example, it was possible to boost the engine RPM temporarily, disturb engine timing, disable all cylinders simultaneously, disable the engine such that it knocks excessively when restarted, or cannot be restarted at all, disable the engine completely and adjust the engine's idle RPM.

Brakes . Using fuzzing, it was possible to know how to control either individual brakes or a set of brakes. It was possible to disable brake while the car is in motion (mounted on a jack), preventing the driver from stopping the car.

HVAC . It was possible to discover how to control HVAC system. In some cases, the effect of the used CAN message overrides any user input.

Disable communications . It was also possible to disable communications from/to individual ECUs like Engine Control Module (ECM) and the Body Control Module (BCM). Disabling communication of ECM resulted in reporting 0 speed of the car even if it is moving. While, disabling communication of BCM resulted in freezing the instrument panel cluster.

1.2.2 Indirect access

All the experiments that have been made using direct access assumed that the adversary already have direct physical access to the OBD II port. This section describes the experiments [4] that have been done using indirect access means.

Indirect access to OBD II Port

During regular service, OBD II port is accessed either via a dedicated device or via a computer (through a PassThru device[2]). In the former case, such device can be programmed by a computer. In both cases, if an adversary could gain access to any of the computers (either the one that programs the device or that is connected directly to the PassThru device), then it is possible to compromise any car that is under service. Such threat can either target all cars that are being serviced or affect a specific car being identified by its Vehicle Identification Number [3].

After doing some analysis, two vulnerabilities were discovered in the PassThru device. First, if an attacker is connected to the same WiFi network of the PassThru device, then he can easily connect to the PassThru device and accordingly control the car that is connected to the device. Second, the attacker may also insert malicious code into the PassThru device. This way, several cars may be attacked using the same device. Also, it was possible to use the PassThru device to spread the malicious code to any other PassThru device in the same WiFi network.

[2]The computer is connected to the PassThru device using USB or WiFi

[3]Since 1981, global automotive manufacturers have utilized a complex numbering system called a Vehicle Identification Number (VIN) that uniquely describes a vehicle. This number provides a coded description of the vehicle including: manufacturer, year of production, place of production and vehicle characteristics [5]. For some cars, the Vehicle Identification Number is located at the bottom of windscreen at the driver side and can be easily read from outside the vehicle

Access through infotainment systems

Car manufactures and suppliers have made many advances in the field of in-vehicle infotainment in order to provide some useful features to the car driver as well the passengers. Two examples of the new technologies are Ford SYNC [6] and BMW iDrive [7]. In general, many infotainment systems in modern cars provide an interface for connection to user devices like iPod and iPhone. Moreover, it is now possible using MirrorLink®(previously known as TerminalMode [8]) to control the data displayed on the car head unit using smart-phone. Typically, such infotainment systems are not isolated from the networks within the car. This is either to display some useful information to the user or to control some devices using software-based HMI (using touchscreen). This introduces a possible attack surface from which the internal network of the car can be accessed.

For the car under test, there was a vulnerability that could reflash the infotainment by simply inserting a CD of a certain image name. The contents of the CD would be flashed to the ECU. This vulnerability could be exploited to inject malicious software into the telematics unit. The second vulnerability was discovered after doing analysis, debugging and reverse engineering to the code responsible for parsing the media files. After the analysis, it was possible to create a WMA file that can be played normally on the player but in the background sends arbitrary messages on the CAN bus.

Short-Range Wireless Access

In modern cars, there are many newly introduced short-range wireless technologies used. Those technologies aim at increasing the safety of the cars together with increasing the comfort of the passengers. However, these channels may provide a back-door for an adversary to gain control over the car. This is a possible scenario if the ECU connected to any of these wireless interfaces has some vulnerabilities. In this case, the adversary may exploit such vulnerabilities to control the car. Here is a list of the currently deployed wireless technologies:

Remote Keyless Entry (RKE) . RKE systems are used to open the door locks remotely and deactivate alarm systems.

Tire Pressure Monitoring Systems . TPMS are used to warn the user of over/under inflation of tires.

RFID-based immobilizers . Immobilizers are used to authenticate the original key of the car. Such systems comprise an RFID reader located inside the car that identifies the RFID tag of the car key and accordingly activate the immobilizer.

WiFi . It may be also used in some modern cars to provide wireless internet connectivity for the passengers.

Bluetooth . It is mainly used for hands-free usage of smart-phones while driving. There are two possible ways to perform an attack using bluetooth; indirect and direct. For indirect attack, the attacker is assumed to have direct access to paired device. It was possible to develop a trojan horse on a smart-phone and use it to send attack payload to the telematics unit. On the other hand, for direct attack, the attacker is not officially authorized to connect to the bluetooth interface. This type of attack is made on two steps. First, the attacker tries to obtain the MAC address of the bluetooth device of the telematics unit. The second step is to brute-force the pin code. Actually, the MAC address can be obtained by sniffing on the messages exchanged while another device is paired. It can be also obtained by sniffing on paired device in the absence of the car.

Long-range wireless communications - Cellular

Modern cars include interfaces for cellular connectivity in order to have voice and data communication. For critical data, the voice channel is used since high speed data connections may not be available in some areas. An example of this case is to call for help after an accident happens. The system was compromised using a laptop that makes several calls to the car until it authenticates, then exploit buffer overflow vulnerabilities in the code of the communication stack to download malicious code from the internet using the 3G data connection. Since, the voice channel is used for data communication, it was possible to encode an audio file with the payload of the post-authentication attack and then dial the car number and play the file resulting in the same effect.

1.2.3 Triggering the attack

After an adversary could inject the malicious code into the car through any of its vulnerable interfaces, the attack can then be triggered either immediately or based on a trigger condition. The trigger condition can depend on the state of the car; for example its speed, its GPS location...etc. However, with the many vulnerable wireless interfaces found in cars, it is possible to control the car remotely and trigger an attack when needed. This allows the synchronization of attacks between many cars at the same time. Moreover, it is also possible to exploit data connectivity feature of the car to monitor the car remotely and manage the attack in a more organized way.

TPMS-based trigger

TPMS was exploited as a trigger method by two ways. In the first way, they reflashed the telematics unit so that it executes an attack when it receives an arbitrary reading from TPMS ECU. In the second way, they reflashed the TPMS ECU so that when it receives specific TPMS packets, it starts to send predefined messages on the CAN bus. In the latter way, the attack was initiated by sending the chosen TPMS packets.

FM RDS trigger

This trigger mechanism was realized by reflashing the telematics unit. The modified code triggers the attack when it receives a particular "Program Service Name" message [9]. When an attack is triggered, the telematics unit sends arbitrary CAN messages.

Cellular trigger

The realization of this trigger also was based on reflashing the telematics unit. The modified code downloads and runs small code from the internet. This code is an IRC client that connects to an IRC server (using the 3G connection of the car) and respond to its commands. It was possible to use that IRC server to control two separate cars.

1.3 Motivation behind attacks

There could be many reasons for which someone might want to perform any of the attacks discussed above.

1.3.1 Theft

The main motivation behind attacking cars is to steal them. As mentioned above, it could be possible for the adversary to unlock the door locks under certain trigger conditions. Moreover, it is possible to track the location of a car by sending its GPS location using 3G connection. Hence, a thieve could locate the victim car and log into it, bypass the immobilizer, start the engine and take it away. This theft mechanism has been realized in [4].

1.3.2 Surveillance

The microphone used for hands-free calling can be used for recording conversations that are being held inside the car. Those recordings can be sent either in real-time or in a later time through a compromised 3G connection. Moreover, the GPS location information of a car can be also tracked in real-time. As a result, if a car is compromised, it would be possible to track it and all the conversations that are being held inside.

1.3.3 Kidnapping

If the adversary can display false traffic information on the head unit, then he could direct the victim towards an alternative road where he get kidnapped. It is also possible to play threatening messages on the speakers of the car together with locking doors and doing any irritating effects like activating the horn, flashing the internal light...etc.

1.3.4 Terrorist attacks

Since the level of attacks can cover a large number of cars, it would be possible for terrorists to do disasters by gaining control over a large numbers of a street in a certain city. For example, the attacker may disable brakes of all controlled car simultaneously causing major accidents. The attacker may also disable the wipers during a day with heavy rain or snow, thus causing many accidents.

1.3.5 Assassination

A famous way of assassination that was used tens of years ago is to disable the brakes of a car. Unfortunately, this type of attack is still valid if the attacker can control the brakes of the car. If the steering of the car is done by wire, then it would be also possible to disable the steering wheel causing severe accidents. The difference between such attacks and the older ones is that old ways always left a trace while it is possible to hide traces for modern attacks. This can be realized if the attack is held by downloading some code in the RAM and then reset the ECU after the accident to remove any trace.

1.4 Reasons of vulnerabilities

1.4.1 Inherent weaknesses

In [2], it has been highlighted that some of the weaknesses of vehicles security arise from the widely used 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 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.

Besides the direct weaknesses of CAN bus listed above, there are also access control weaknesses due to the current used protocols.

- Reflashing: In order to protect an ECU from being reflashed by an unauthorized party, a challenge-response protocol is used. By design a service shop can do such authentication in order to upgrade the software of the ECU.

- Testing: For diagnostics, there exists a service that allows the tester in the workshop to control some vehicle parts directly (skipping the user input). Such service is protected by a challenge-response pair.

The main drawback in using this protocol, is that for each ECU, this challenge-response pair is fixed. This is to avoid having the algorithm being run inside the ECU. With limited length of key/seed pairs, it could be possible to determine the key in about 7 days!

1.4.2 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.

- Disabling communications: It is specified that a car shall reject any command that disables communication as long as the car is moving. However, this rule is not always respected. Thus an attacker, can disable communications while the car is moving leading to severe accidents.

- Reflashing while driving: It is also mandated that reflashing the code of an ECU is not allowed while the car is moving. However, it was possible to reflash some ECUs while the car was moving.

- Reflashing the telematics unit. It has been found that the telematics unit of the car under test had a single seed-key pair that is used for all ECUs deployed in all cars. Moreover, it is also possible to reflash the ECU even if the wrong key is entered.

- Unrestricted access to keys: In order not to run the challenge-response protocol in each ECU, manufacturers store the keys inside the ECUs themselves. The area in memory where those keys are stored should have restricted access. However, it was possible to read those keys using “memory read” service.

- Same keys used for both reflashing and DeviceControl: It has been also discovered that some ECUs use the same key for authenticating both DeviceControl and reflashing. As a result, when a key is known, the access to both services is open.

- Reflashing gateways from low-speed network: Gateways are used to bridge different CAN networks. Networks are of two types; high speed and low speed. High speed are considered more trusted than low speed. As a result, standards specify that a gateway can be reflashed only from the high speed network. However, it has been discovered that it can be also reflashed from the low speed network.

With all these vulnerabilities, there is a need for several studies to secure in-vehicle networks.

1.5 Organization of the thesis

The rest of the thesis is organized as follows. Chapter 2 discusses the work that has been previously done in securing in-vehicle networks. Then, chapter 3 describes the threat model and identify security requirements. Following that, we introduce different levels of security and propose a new authentication protocol for in-vehicle networks. In chapter 4, we provide a comparison between the proposed protocol and other existing protocols. Later in chapter 5 , we identify future work towards enhancing the security of in-vehicle networks and provide a conclusion for the thesis.

Chapter 2

Related Work

2.1 EVITA Project

To the best of my knowledge, the largest project that aimed at securing in-vehicle communication networks is the EVITA project [10]. EVITA stands for E-safety Vehicle Intrusion proTected Applications. The project took place in the period from July 2008 to December 2011. The project was co-funded by the European Commission. The objectives of the project were to design, to verify, and to prototype an architecture for automotive on-board networks where security-relevant components are protected against tampering and sensitive data are protected against compromise. Thus, 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 communications. 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.

The EVITA project has inferred the following set of security requirements and related functional requirements in order to satisfy the stated security objectives [11]:

- Integrity/authenticity of e-safety related data: Actions depending on critical information should be decided based on assurances about integrity and authenticity in terms of origin, content, and time. Forgery of, tampering with, or replay of such information should at least be detectable.
- Integrity / authenticity of ECU / firmware installation / configuration:

Any replacement or addition of an ECU and/or its firmware or configuration to the vehicle shall be authentic in terms of origin, content, and time. In particular, the upload of new security algorithms, security credentials, or authorizations should be protected.

- Secure execution environment: Compromises to ECUs should not result in system wide attacks, primarily with regard to e-safety applications. Successful ECU attacks should have limited consequences on separate and/or more trusted zones of the platform.

- Vehicular access control: Access to vehicular data and functions should be controlled (e.g. for diagnosis, resources, etc.)

- Trusted on-board platform: The integrity and authenticity of operated software shall be ensured. An altered platform might be prevented from running in an untrusted configuration (e.g. via comparison with a trusted reference) if so required.

- Secure in-vehicle data storage: Applications should be able to use functionality in order to ensure access control to as well as the integrity, freshness and confidentiality of data stored within a vehicle, especially for personal information and security credentials.

- Confidentiality of certain on-board and external communication: The confidentiality of existing software/firmware as well as updates and security credentials shall be ensured. Some applications might additionally require that part of the traffic they receive or send internally or externally should remain confidential.

- Privacy: A privacy policy shall be enforceable on personal data stored within a vehicle or contained in messages sent from a vehicle to the outside. For example, some applications should limit the ability to link sent messages.

- Interference of security functionality: The operation of security services must not negatively affect the availability of bus systems, CPUs, RAM, or of the radio medium.

2.1.1 Security Module

In [12], Marko Wolf et al introduced the idea of using a *security module* for providing different cryptographic functionalities to vehicles. Both centralized and distributed approaches are discussed. In the centralized approach, a single security module is used for providing security to several ECUs within the car. On the other hand, the distributed approach is 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.

2.1.2 Key Distribution Protocol for CAN

In [13], a key distribution protocol has been introduced for securing in-vehicle communications over CAN bus. Due to the embedded constraints, symmetric key cryptography was used. In each ECU, a hardware security module (HSM) was attached. The HSM implements some cryptographic primitives as well as securing the key storage. Moreover, it stores meta-data for the keys (called user-flags). For example, a key may be tagged for signing at a node while it is tagged for verifying at another node.

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 n 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.

In order to be able to send messages larger than the standard CAN payload of 8 bytes, segmentation of data had to be used. The standard ISO 15765-2 [14] which is already used for diagnostics employs segmentation. However, the protocol has been enhanced in order to add security header.

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.

2.1.3 Conclusion

EVITA project has made a considerable progress in securing in-vehicle communications. To achieve this, the concept of adding a hardware security module (HSM) was introduced. The main disadvantage of using HSM is that it infers additional hardware cost to be added to the cost of manufactured vehicles.

2.2 Message Authentication Protocol over CAN

2.2.1 Overview

The problems associated with implementing a backward compatible message authentication protocol on the CAN bus has been discussed in [15]. The CAN bus - since its invention in 1986 by Robert Bosch GmbH - had its design being focused on safety. As a result, it has no built-in ways for inferring security. This lead to the demonstration of several means of attacks such as controlling brakes [2]. Authentication protocol requirements listed in [15] are:

- **Message authentication**
  It is required to make sure that the message has been sent from a trusted node.

- **Replay attack resistance**
  Re-sending a previously sent message should lead to the message being discarded by all its receivers.

- **Group keys**
  It is possible to use the same key for authenticating a group of messages.

- **Backward compatibility**
  Nodes supporting authentication can co-exist with old nodes not supporting authentication.

The first requirement could be met by attaching a message authentication code (MAC) to a message. However, due to hard real time constraints on CAN messages, the used algorithm for generating the code and verifying it need to be fast. The algorithm proposed in the paper is HMAC [15] requesting the hash function used to be fast.

The second requirement could be met by inserting a counter value inside MAC calculations. The strength of this method will depend on the length of the counter. The larger the length of the counter, the less probable it is used more than once in a considerable amount of time.

The most challenging requirement is the last one. This is because adding any extra data to a message will 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+ [16]. Using CAN+, additional data bits can be inserted within the transmission period of each CAN bit. As shown in figure 2.1, CAN+ bits can be inserted in the zone between the “synchronization zone” and the “sampling zone” without disturbing the normal transmission of CAN bits.

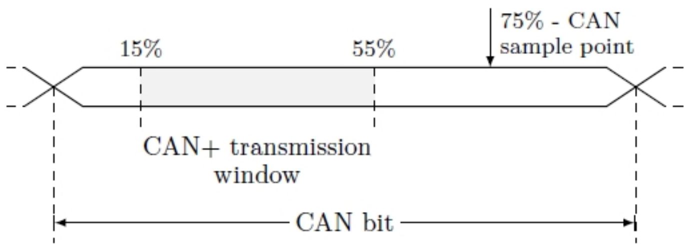


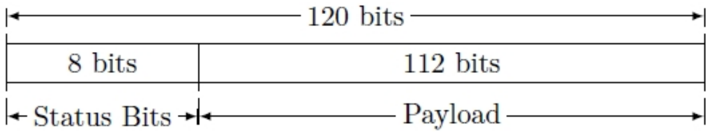
Figure 2.1: CAN bit Timing

As reported by Ziermann et al [16], the number of extra bits to be transmitted depends on the ratio between the frequency of CAN+ bits to regular CAN bits according to the equation 2.1.

$$\frac{CAN + databits}{CANdatabit} = \frac{1MHz}{fbus} * (16 - 1) \tag{2.1}$$

The length of authentication data that is sent using CAN+ is determined based on the shortest CAN message i.e. a message of 1 byte of data. Given the equation above and assuming the same frequency of regular CAN, then the length of authentication data is 15 bytes (120 bits). Those 120 bits are divided into 2 parts:

- 8 Bits: Status Bits
- 112 Bits: Payload



2.2.2 Authentication Protocol

The authentication protocol starts with the establishment of a session key for each group of messages. The length of this key is 128 bits. This step assumes that all receiving nodes have a pre-shared key for this group of messages. It is also assumed that those keys are stored securely within each node. The node sending the group of messages (in case all messages of the group are sent by the same node) or one of the senders (in case not all the messages are sent by the same node) initiates the key establishment process. In this step, the responsible node sends a counter value together with a random number encoded as CAN+ data as shown in figure 2.2.

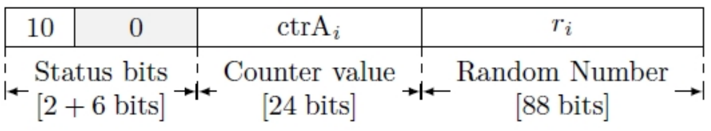


Figure 2.2: CANAuth Key Establishment

Each of the receiving nodes, will apply HMAC using the preshared key to the counter and the random number. The session key is calculated from the result of this step as in equation 2.2:

$$K_{s_i} = \mathrm{HMAC}(K_{p_i}, \mathrm{ctr}A_i \parallel r_i) \mathrm{mod} 2^{128}. \tag{2.2}$$

The purpose of the counter value used here is to prevent replay attacks. In such attacks, the adversary may try to resend a previously sent key establishment message. To prevent this scenario, each of the receiving nodes stores the last received counter value in non-volatile memory. The node does not accept key establishment unless the received counter value is greater than the last used value. This puts limitation on the lifetime of the protocol since the counter value has a limited length. However, with 24-bits of length, this allows one key establishment per minute for 32 years.

In the second part of key establishment, the initiating node sends a signature as shown in figure 2.3.

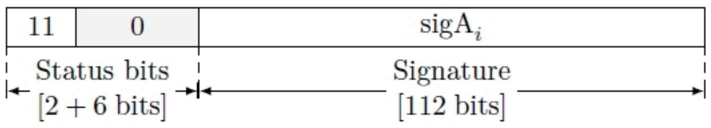


Figure 2.3: CANAuth Key Establishment Signature

where the signature is calculated using equation:

$$\text{sig}A_i = \text{HMAC}(K_{s_i}, \text{ctr}A_i \parallel r_i)\text{mod}2^{112} \tag{2.3}$$

By this step, all nodes verify that the initiating node have the same session key.

In runtime, the authentication of messages goes as follows.

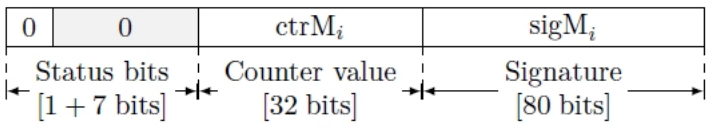


Figure 2.4: CANAuth Runtime authentication

where the signature is calculated using the equation:

$$\text{sig}M_i = \text{HMAC}(K_{s_i}, \text{ctr}M_i \parallel msg_i)\text{mod}2^{80}$$

As shown in the figure 2.4, 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. In a regular CAN bus, any node can send an error frame at any time and thus invalidating the message that is being transmitted. For the management of such errors, each node stores an error counter. When the node sends an error frame, it increments the counter by 8. When the node receives an error frame, it increments the counter by 1. However, a node can send an error frame only when its counter is less than 127. This protects messages sent on the bus from being invalidated by a faulty node. Upon each successful message, the counter of each node is decremented by 1. In CANAuth [15], when a node cannot authenticate a message, it discards the message and sends an error message. As a result, all other nodes discard the message even if they have successfully authenticated the message.

2.2.3 Security Analysis

Adversary Model

It is assumed that the adversary has access to the CAN bus and all messages transmitted on it. Hence, he can easily make a man-in-the-middle attack. However, it is assumed that the adversary has no access to the pre-shared keys stored within each node.

Denial of Service attacks

During the key establishment phase, the adversary may alter the message that is being sent. This results in generating two different sessions keys. Using the protocol, this can be easily detected using the 2^{nd} message of this phase. Thus, the nodes will retry to establish a new session key and so on.

2.2.4 Conclusion

The idea of using CAN+ to send authentication data does not affect the communication bandwidth since data is sent out-of-band. This means that it does not require any modifications to be done to the existing CAN messages sets. However, it needs to use a modified physical layer. Therefore, it cannot be used with existing CAN controllers and tranceivers.

2.3 CAN message encryption using AES and attacking it using CPA

2.3.1 Overview

In [17], confidentiality has been added to CAN protocol using AES symmetric encryption. The breaking of this algorithm has been demonstrated using side-channel analysis.

2.3.2 Encryption

In order not to affect hard real time constraints of automotive systems, it is decided to encrypt only selected CAN messages. The parts of CAN messages that need to be encrypted are the ID (11-bits), DLC (4-bits) and the data (up to 8 bytes). Thus the length of plaintext is 10 bytes. However, the block size of AES is 128 bits (16 bytes). Hence, padding is added to the 10 bytes of plaintext before the encryption is done. The result 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.

2.3.3 Correlation Power Analysis

Correlation Power Analysis (CPA) is based on the relation between the power consumption and the Hamming weight of the data being processed. Typically, the difference in power consumption between a set of data and another is too small. However, using statistics it is possible to differentiate such differences and compromise the system.

2.4 Multiple MAC Per Receiver

Using symmetric cryptography for multicast authentication has been introduced in [18] and [19]. In those papers, 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 [18] [19] 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.

2.5 The TESLA protocol and its variations

TESLA protocol was proposed in [20] 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 [21].

In order to achieve immediate authentication, a modification to TESLA protocol was proposed in [22]. In the proposal, messages do not need to be queued at the receiver waiting to be 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 [23] 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 for Secure Real-time Transport Protocol (SRTP) as in [24]. Recently, another modification was made for TESLA introducing TESLA++ [25]. 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.

2.6 Conclusion

Several CAN authentication protocols have been proposed before. Some of them were based on hardware while others were based on software. Using hardware is more efficient either from point of view of security strength. However, it requires additional cost to be added to each ECU that is being manufactured.

Using software looks more promising if it can provide the required security strength. Therefore, it is required to design an efficient CAN authentication protocol that can be implemented by software.

Chapter 3

Security Levels and Authentication Protocol

3.1 Threat Model

In-vehicle networks consist of various ECUs that are interconnected using communication buses. There are many types of communications buses used like CAN [26], LIN [27], FlexRay [28] and MOST [29]. CAN is the most widely used type of buses. As discussed earlier, CAN messages do not contain any source or destination addresses. Also, they do not provide any means of authentication. Hence, any adversary node that succeeds to have access to the bus can listen to any transmitted message. Moreover, it can inject malicious messages into the network.

In the following sections, we will discuss a multilevel approach for securing in-vehicle networks. Then, 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 modified to be used in other communication bus types.

3.2 Levels of Protection

In order to protect vehicles from various attacks, we recommend three levels of protection:

- Level 1: Prevent ECUs from being compromised

- Level 2: Protect the vehicle's internal network against a compromised ECU
- Level 3: Detect the compromise of an ECU

3.2.1 Level 1: Prevent ECUs from being compromised

The first level of protection is to protect every ECU inside the vehicle from being compromised. There are many ways by which an ECU can be compromised. Here is a list of some of them:

- **Flash Bootloader.** An ECU can be compromised by replacing the flashed software by another malicious software using the flash bootloader. Typically, the flash bootloader is a piece of software that facilitates the update of the software flashed on an ECU. It is used by car manufacturers to update the software of ECUs when necessary. Basically, accessing the bootloader is secured by challenge-response pair. To protect the ECU, the value of the key used in the authentication algorithm shall be unique for each ECU. Also, the length of the key used should be large enough in order to be protected against brute-force attacks. The value of the key stored in the flash memory of the microcontroller shall be secured against malicious reading. In order to protect the ECU from brute-force attacks, if the wrong response is entered for few times, the ECU shall be locked and cannot enter bootloader mode anymore.
- **Flasher.** In this case, the adversary has direct physical access to the ECU. The protection against such attack can be done by censoring the flash memory during the production. By that way, the bootloader is the only entity that is granted the access to flash the ECU. It is worth noting that this type of attack is special because the attacker who can connect a flasher to the ECU can replace the microcontroller itself or even the whole ECU rather than replacing the software.
- **Exploiting vulnerabilities.** When the software of the ECU contains some vulnerabilities, they can be used by an attacker to insert his malicious code. Several types of vulnerabilities exist. Among these are the buffer overflow. The way of protection for such attacks is to perform

static analysis to the software to ensure that the software is free from such vulnerabilities.

3.2.2 Level 2: Protect the vehicle's internal network against a compromised ECU

In the second level of protection, we assume that at least one ECU was compromised by an adversary. The compromised ECU can be one of two cases:

- **Case A:** An ECU which is part of the vehicle's internal network is compromised (using any of the ways mentioned in Level 1 protection).
- **Case B:** Any malicious node that is connected to the bus by the adversary.

The goal of protection in this level is to prevent those compromised ECUs from harming the vehicle's internal communication network. The compromised ECU can harm the vehicle's network by doing any of the following actions:

1. Send a malicious message on behalf of another node.
   This type of attack arises from the nature of the CAN bus since it does not specify methods of source authentication. As a result, the compromised ECU may send false messages on behalf of other ECUs. This can be protected by using a source authentication protocol in the communication in the CAN bus.
2. Send a malicious message that it normally sends.
   Unfortunately, this type of attack cannot be protected after the ECU is already compromised. This is because whatever cryptographic methods used, the compromised software can still use this secured communication stack to send messages containing false data.
3. Perform a denial of service attack on the CAN bus. This can be protected by using something like the bus guardian [30] of the FlexRay protocol.

3.2.3 Level 3: Detect the compromise of an ECU

As a 3^{rd} level of protection, it should be possible to detect the compromise of any node of the vehicle's internal network. The process of checking ECUs can either be done regularly by the vehicle or at least be done during regular service of the vehicle. The proposed way is to verify the checksum of the flash contents of each ECU. This can be the responsibility of a new Security ECU or the body controller. When the software of any ECU is updated, then the new checksum must be communicated to the Security ECU.

3.3 Security requirements

Levels 1 and 3 can be viewed as recommendations that can be adopted directly by vehicles' manufacturers. However, level 2 needs some research. We will focus on solving the problems mentioned above in Level 2. In this case, it is required to protect the vehicle's internal network from a compromised node. We will focus on the two cases A & B:

Case A: The threat model of this case is as follows. The CAN bus has some ECUs connected to it. One of these ECUs was previously compromised by an adversary.

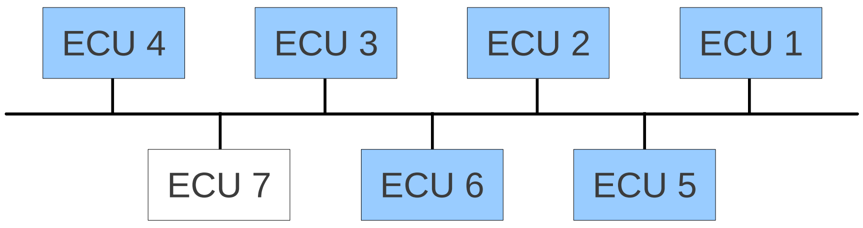


Figure 3.1: Case A: Existing ECU is compromised

Case B: The threat model of this case is as follows. 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.

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.

The following requirements are needed when designing the authentication protocol.

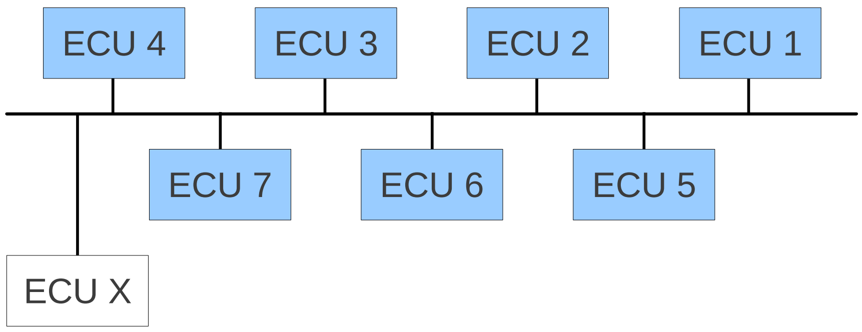


Figure 3.2: Case B: Adversary node connected to the bus

1. The message shall contain an evidence that can be *generated* only by its trusted sender.

2. The receiver shall be able to *verify* that evidence.

3. The receiver shall not be able to *re-transmit* the message masquerading the trusted sender.

4. 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.

5. The protocol shall not require either heavy computation or high memory consumption. This is because the currently produced ECUs use microcontrollers with limited resources.

3.3.1 Basics of authentication

Before discussing multicast authentication, we introduce a brief discussion about unicast authentication. Message source authentication between a sender and a single receiver can be achieved by adding a message authentication code (MAC) to each transmitted message. The sender calculates the MAC based on the message contents and a shared secret. On the other side, the receiver uses the same shared secret to verify the MAC. This can be achieved using symmetric cryptography.

When there is more than one receiver - the case of multicast communication - another way shall be used. The shared secret cannot be shared between more than two entities. Therefore, asymmetric cryptography is needed. For each transmitted message, the sender creates a MAC using its private key. Various receivers use the sender public key to verify the MAC. However, asymmetric cryptography is not recommended for our domain because it needs high computational capabilities.

TESLA is one of the most famous authentication protocols for multicast communications. However, if we try to adopt it in CAN networks we will find that we will need large communication overhead. This overhead is 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.

3.4 The Proposed CAN Authentication Protocol

3.4.1 Protocol Overview

In this section, we propose a lightweight authentication protocol to be used in CAN networks. The protocol is designed to satisfy the requirements described in previous section.

CAN Frame Format

The CAN message data frame format takes the following form:

The length of the “Data Field” (which carries the actual payload of the message) varies from 1 byte to a maximum of 8 bytes.

Regarding the “Arbitration Field”, it takes one of two forms as shown in the figure below:

- Standard Format: The arbitration field consists of 11 bits representing the message “Identifier” in addition to the RTR (Remote Transmission Request) bit.

- Extended Format: The arbitration field consists of 29 bits representing the “Identifier” in addition to SRR (Substitute Remote Request), IDE (Identifier Extension) and RTR (Remote Transmission Request) bits.

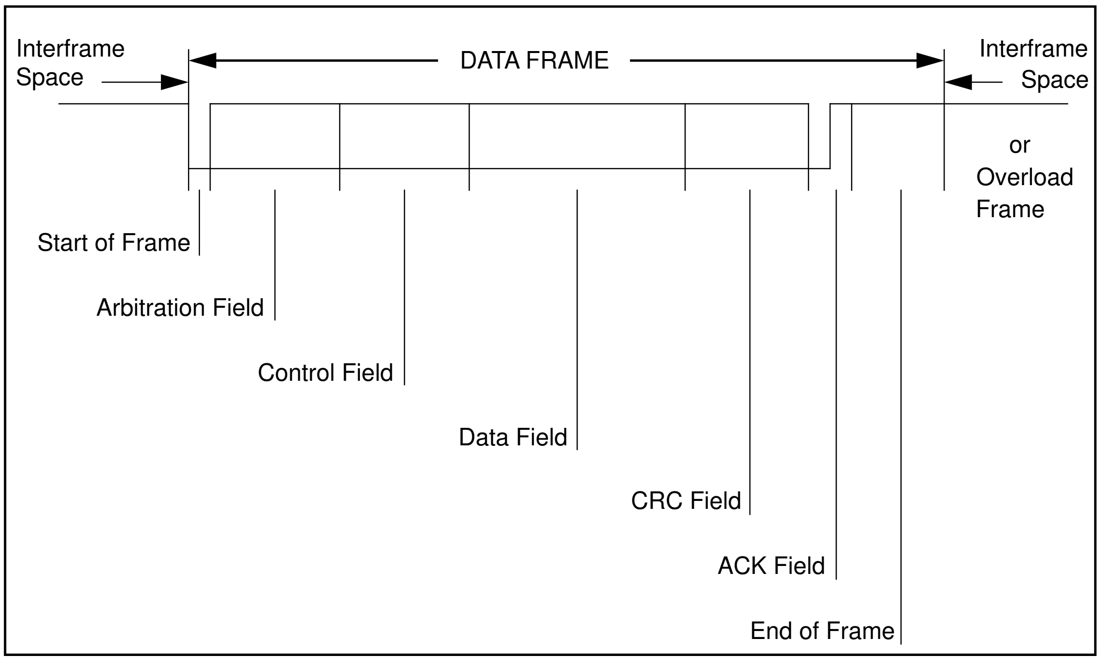


Figure 3.3: CAN Data Frame Format

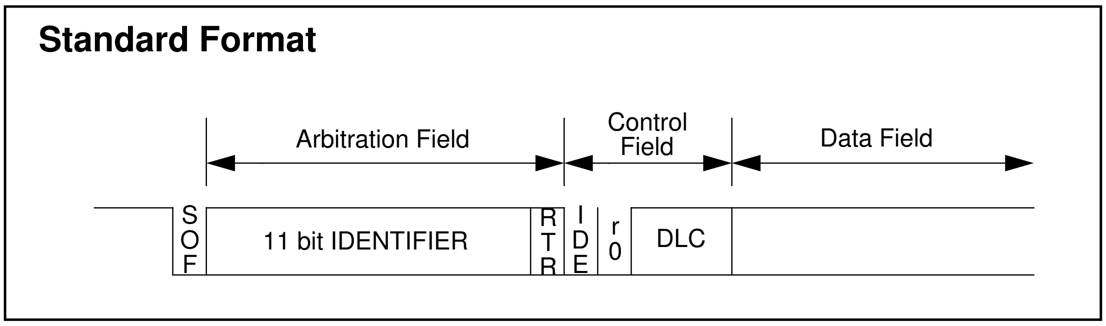


Figure 3.4: CAN Standard Arbitration Field

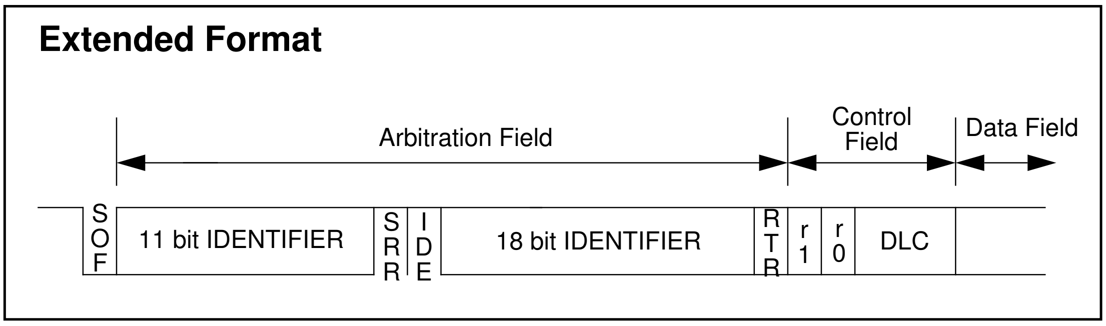


Figure 3.5: CAN Extended Arbitration Field

Main Idea

From our point of view, it is only required to append an authenticator to the message that can be verified by the receiver. This authenticator can be only selected by the sender and verified by the receiver. As a first step, we assume that the authenticator can be a “magic number” that can be generated using a one-way hash function like that 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 through a secure channel 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.

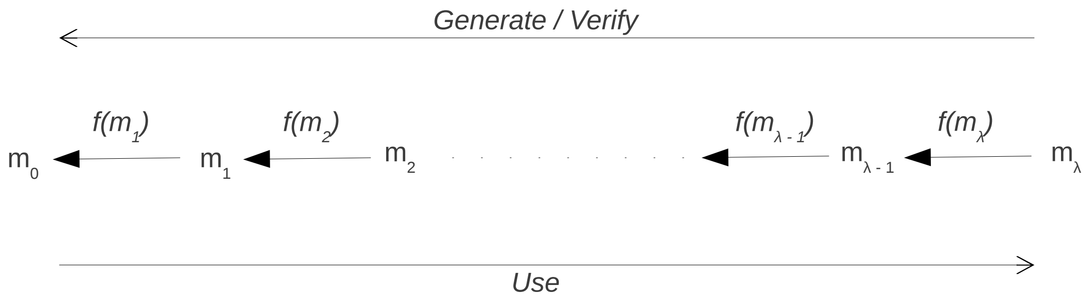


Figure 3.6: Magic Number Chain

However, since the magic number does not depend on the data that is being sent, then the protocol could be attacked using the following scenario. An attacker may listen to the authenticated message and capture the magic number part and then corrupts the bus to stop the message while being transmitted and then sends a false message using the same magic number. As a result, it is not enough to send the magic number alone. The authenticator shall be function of the data that is being transmitted as well. Therefore, we choose the authenticator to be the result of XOR operation between the magic number and the hash of the data. On the other side, the receiver shall calculate the hash of the data, then XOR it with the received authenticator to obtain the magic number. Then, it shall verify the magic number.

Modes of Operation

The protocol can be used in one of two modes:

- Extended Mode

- Standard Mode

In the *Extended Mode*, the authenticator 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 authenticator. 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 authenticator 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.

Message losses dues to electromagnetic interference is taken into consideration. There are two types of transmitted messages; periodic messages and event-driven messages. For periodic messages, the receiver can detect how many messages are lost and hence can recover easily. However, for event-driven messages, when the receiver fails to verify the magic number versus the last received message, it performs some transformation till the magic number is verified. The maximum number of trials is a configuration parameter for the protocol.

Since the length of the magic number is small; 16 bits only then we shall consider brute-force attacks. Therefore, the magic number chain has to be updated frequently.

3.4.2 Protocol Details

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.

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.
- $h(x)$: Hash function applied on a variable x.

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.

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 3.7, 3.8 and 3.9. The horizontal axis represents the bits while the vertical axis represents the bytes. In figure 3.7, the first two bytes are not used. The next four bytes contain a nonce. The last two bytes contain a checksum. In figures 3.8 and 3.9, the first two bytes are used to carry an authenticator. For all types of messages, 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.

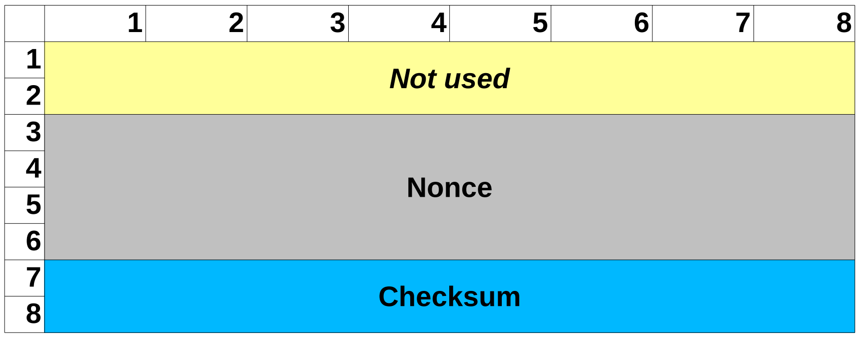


Figure 3.7: (Channel Setup) / (Soft Sync) / (Hard Sync) Request Message

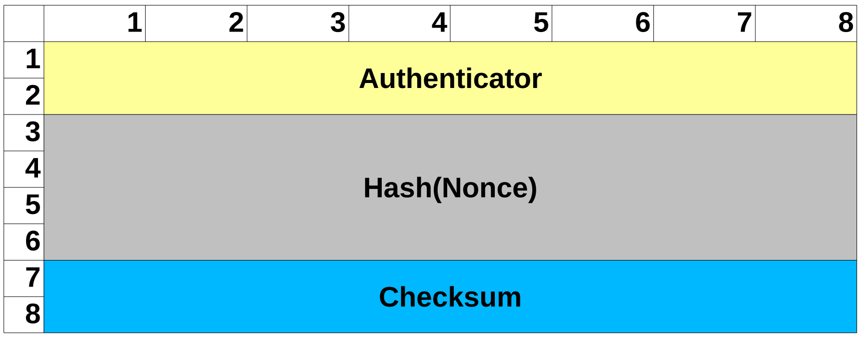


Figure 3.8: First Response Message

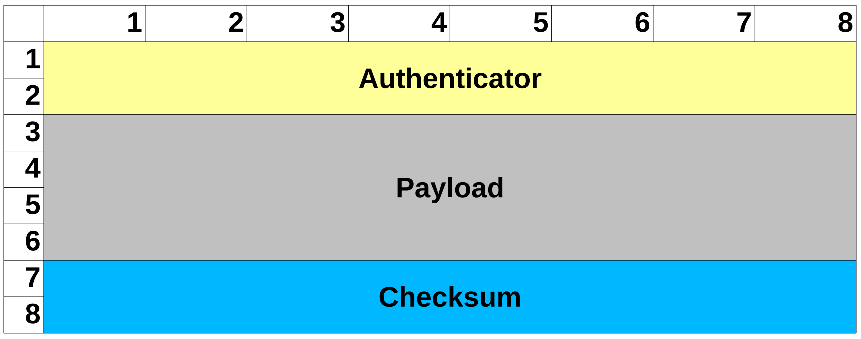


Figure 3.9: Consecutive Response Message

3.4.3 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

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.

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 an "authenticator" 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.

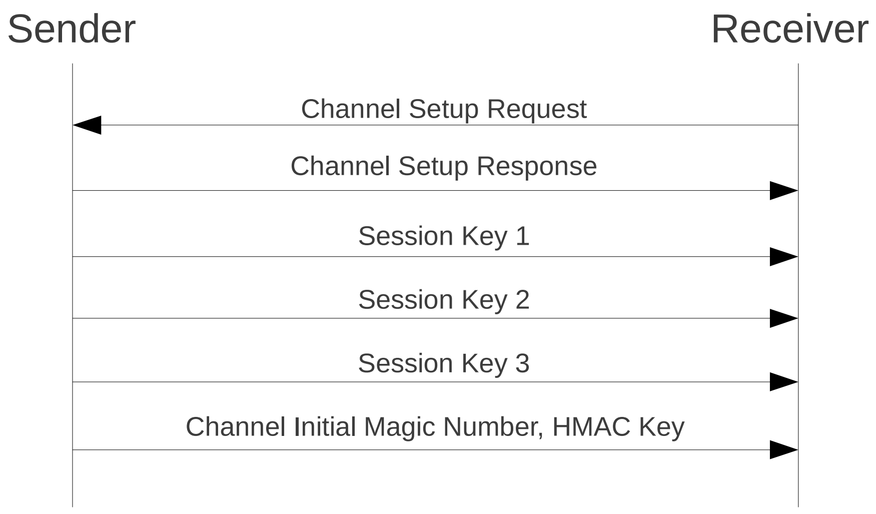


Figure 3.10: Channel Setup

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 [31], 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.

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.

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 generates the authenticator by XORing the current magic number M_{ij} with the hash of the current message D_{ij}.
2. The sender appends the authenticator to the current message.
3. The sender encrypts the resultant using the session key K_S.

The sent message takes the form:

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

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 authenticator.
3. The receiver extracts the magic number M_{ij} by XORing the hash of the data and the authenticator.
4. 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.

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 [31], 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}.

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.

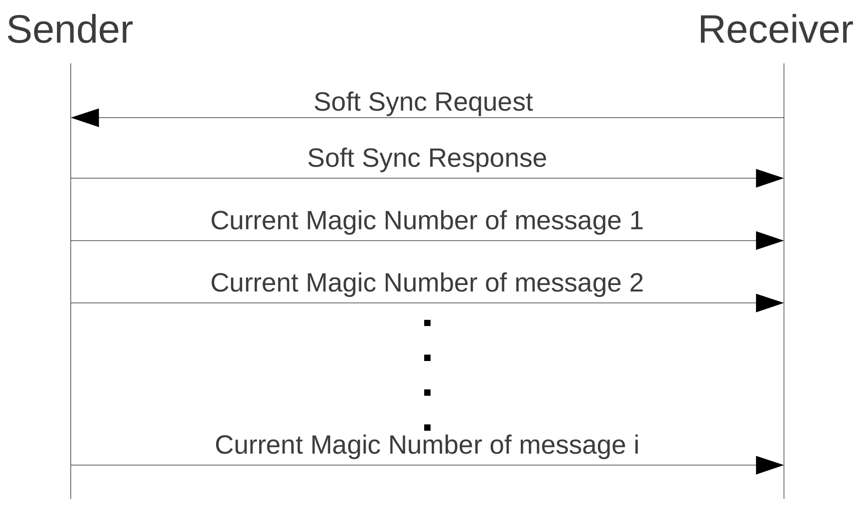

Sender
Receiver
Soft Sync Request
Soft Sync Response
Current Magic Number of message 1
Current Magic Number of message 2
Current Magic Number of message i

Figure 3.11: Soft Synchronization

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.

3.5 Cryptographic Primitives

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

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

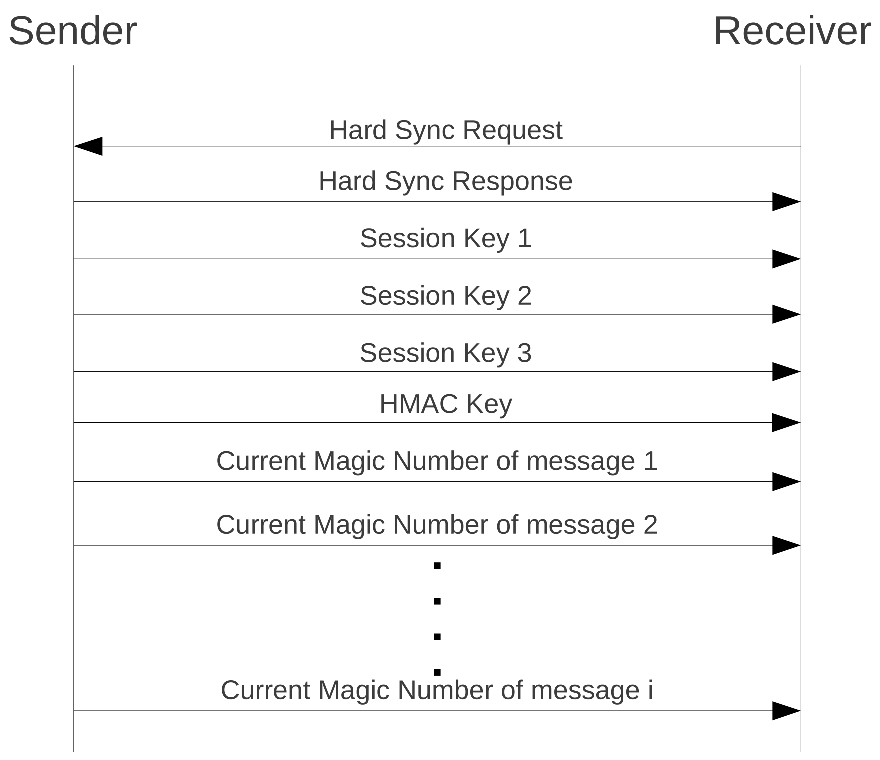

Sender
Receiver
Hard Sync Request
Hard Sync Response
Session Key 1
Session Key 2
Session Key 3
HMAC Key
Current Magic Number of message 1
Current Magic Number of message 2
Current Magic Number of message i

Figure 3.12: Hard Synchronization

3.5.1 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. 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 [32]. However, when the used cipher requires key size larger than 10 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 (same as initial session key). The other solution is to increase the number of messages exchanged in the starting phase. For simplicity, we use RC4 to implement the protocol.

Overview of RC4

RC4 is a simple stream cipher with a key length from 1 to 256 bytes [33]. The key is used to initialize a 256-byte state vector “S”. At any point of time, the state vector contains a permutation of all 8-bit numbers from 0 to 255. The state vector is used to generate the key stream in a systematic manner. The values of the state vector is continuously permuted to generate the stream.

3.5.2 One-Way Hash Function

One-Way hash function is used to generate the chain of the magic numbers. In general, any cryptographic hash function shall have three properties [31]:

- Collision resistance: It is computationally infeasible to find two different inputs to the cryptographic hash function that have the same hash value. That is, if hash is a cryptographic hash function, it is computationally infeasible to find two different inputs x and x' for which $hash(x) = hash(x')$.
- Preimage resistance (One Wayness): Given a randomly chosen hash value, $hash\_value$, it is computationally infeasible to find an x so that $hash(x) = hash\_value$.
- Second preimage resistance: It is computationally infeasible to find a second input that has the same hash value as any other specified input. That is, given an input x, it is computationally infeasible to find a second input x' that is different from x, such that $hash(x) = hash(x')$.

The most important property for our application is the "Preimage Resistance" or "One Wayness". This is to prevent any attacker from generating the chain in a reverse order and hence send false authentic messages on behalf of the actual transmitter. The other property to consider is the "Collision Resistance". Although there is no clear attack that could make use of collision, there is still a need for collision resistance. This is to make the generated chain having unique numbers. If a function has high collision probability (low collision resistance) it is possible to have the generated chain with repeated numbers as shown in figure 3.13:

$$X_0 \rightarrow X_1 ...... X_{23} \rightarrow X_{24} \rightarrow X_{25} .... X_{45} \rightarrow X_{23} \rightarrow X_{24} ....$$

Figure 3.13: Repeating Magic Number Chain

However, if we use a hash function only (for example 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 or record the values of a previously created chain then he could be able to send messages on behalf of the actual transmitter. In order to solve this problem, we choose to use Keyed-Hash Message Authentication Code (HMAC) [34] instead. HMAC requires the use of a cryptographic hash function in conjunction with a secret key. The secret key shall be changed every session. As a result, the values of the magic number chain will vary from a session to another according to the value of the key.

Selection of hash function

As indicated above, HMAC requires the use of a cryptographic hash function. There are many available hash functions from which we can select the one to use for HMAC. The selection criteria depends on two main points:

- The selected hash function shall have high preimage resistance as well as a considerable collision resistance.
- The selected hash function shall add minimum overhead from point of view of execution time and memory consumption (both RAM and ROM).

Initially, we will list the available hash functions and then select which one to use.

- **MD5**

  The first function to consider is MD5 which is defined by RFC 1321 [35]. Its output consists of 128 bits. It is widely mainly used to check data integrity. However, many successful attacks have been discovered for both collision resistance (with a complexity of $2^{20.96}$) [36] and preimage resistance (with a complexity of $2^{123.4}$) [37] [38].

- **SHA-1 and SHA-2**

  The Secure Hash Standard (SHS) [39] specifies 5 hash functions that are approved by the National Institute of Standards and Technology (NIST). The functions are: SHA-1, SHA-224, SHA-256, SHA-384 and SHA-512. The output length in bits of them is 160, 224, 256, 384 and 512 respectively.

- **SHA-3**

  In 2007, NIST announced [40] a public competetion to develop a new hash algorithm. The competition was NIST's response to advances in the cryptanalysis of hash algorithms. The winning algorithm will be named "SHA-3" for SHA 3. According to [41] NIST received 64 entries by October 31, 2008; and selected 51 candidate algorithms to advance to the first round on December 10, 2008, and fourteen to advance to the second round on July 24, 2009. Based on the public feedback and internal reviews of the second-round candidates, NIST selected five SHA-3 finalists - BLAKE, Grøstl, JH, Keccak, and Skein to advance to the third (and final) round of the competition on December 9, 2010, which ended the second round of the competition. In October 2012, Keccak was announced to be the winner algorithm.

Table 3.1 summarizes the output size of different available hash functions together with the strength of each one of them. The values of SHA-2 are obtained from [31].

Table 3.1: Output size and Strength of Hash Functions

Function	Output Size in bits	Collision Resistance Strength in bits	Preimage Resistance Strength in bits	2nd Preimage Resistance Strength in bits
MD5	128	20.96	123.4	N/A
SHA-1	160	< 80	160	105-160
SHA-224	224	112	224	201-224
SHA-256	256	128	256	201-256
SHA-384	384	192	384	384
SHA-512	512	256	512	394-512

Truncation

Regardless the chosen hash function to be used for HMAC, we need to truncate the output of HMAC into 2 bytes only in order to fit for the chosen magic number size. According to [31], the result of HMAC shall not be truncated to less than 8 bits. Since we chose 16-bits, then we conform with this recommendation. Also, [31] states that when truncation is to be done, then the leftmost bits shall be selected. Therefore, we choose to select the 16 leftmost bits of the HMAC output.

Security of HMAC

The security strength of the HMAC algorithm [31] is the minimum of the security strength of K and the value of 2L (i.e., security strength = min(security strength of K, 2L)). For example, if the security strength of K is 128 bits, and SHA-1 is used, then the security strength of the HMAC algorithm is 128 bits. The HMAC key K shall be generated with a security strength that meets or exceeds the desired security strength of the HMAC application, and the approved hash algorithm in the HMAC application shall have a message digest length of at least half of the desired security strength (in bits) of the HMAC application. For example, if the desired security strength of the HMAC application is 256 bits, the HMAC key K shall be generated with a security strength of at least 256 bits, and an approved hash function with the message digest length of at least 256/2 (128) bits shall be used.

Security of HMAC values

The successful verification of a MacTag does not completely guarantee that the accompanying text is authentic; there is a slight chance that an adversary with no knowledge of the HMAC key, K, can present a (MacTag, text) pair that will pass the verification procedure. From the perspective of an adversary that does not know the HMAC key K (i.e., the adversary is not among the community of users that share the key), the security strength provided by a MacTag depends on its length. The length of a MacTag shall be sufficiently long to prevent false acceptance of forged data. For most applications, a length of 64 to 96 bits is sufficient. Shorter MacTags may also be acceptable if the rate of false acceptances does not create a significant impact for the application. For example, in video/audio stream applications, accepting one bad data package in 28 data packages may not create a huge impact for the application.

Computation Overhead

In order to select the best hash function from point of view of computational overhead (processing and memory), two steps are done:

- Checking benchmarks results for different hash functions.
- Experimental results on our target platform.

For the first part, we used two main sources for benchmark results. The first one is the eBASH project [42]. eBASH stands for (ECRYPT Benchmarking of All Submitted Hashes). It is a project in ECRYPT's VAMPIRE lab to measure the performance of hash functions but not target embedded platforms. However, its results give some indication about different hash functions.

The second source is the eXternal Benchmarking eXtension (XBX) [43]. It is an extension to SUPERCOP [1]. It has been successfully used to benchmark many different hash functions on several different microcontrollers.

3.5.3 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".

3.6 Authentication Protocol Implementation

3.6.1 Hardware Platform

The authentication protocol was implemented on StarterTRAK TRK-MPC5604B board manufactured by Freescale Semiconductors. The board contains MPC5604B PowerPC microcontroller. This microcontroller belongs to The Qorivva MPC560xB/C family of 32-bit microcontrollers. This family is intended for use in automotive body electronics applications.

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

[1]SUPERCOP [44] is a toolkit developed by the ECRYPT VAMPIRE lab for measuring the performance of cryptographic software. SUPERCOP stands for System for Unified Performance Evaluation Related to Cryptographic Operations and Primitives

- Number of LIN/UART controllers: 4

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

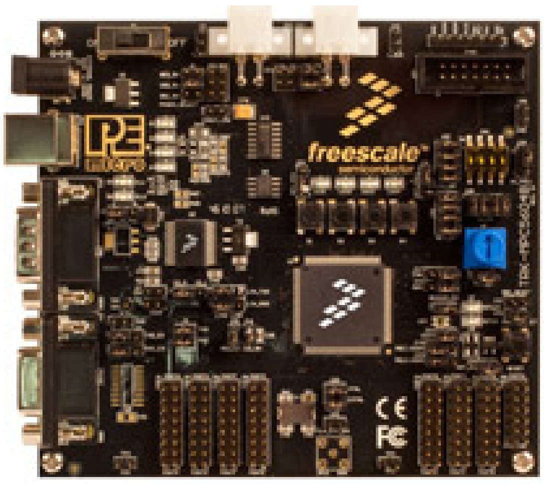


Figure 3.14: TRK-MPC5604B Board

3.6.2 Cryptographic Primitives

As described in the previous section, the authentication protocol relies on some cryptographic primitives. We used an open source library called PolarSSL [45]. It is distributed under the GNU Public License Version 2.0 (GPL v2.0) and any later version of this License. We used the latest available version; version 1.1.4. The library is written in ANSI C targeting embedded systems.

3.6.3 Software Architecture

Dynamic architecture

The real-time behavior of the software is achieved using a simple rate-monotonic scheduler. The scheduler triggers the execution of different tasks according to their statically specified periods. The scheduler is based on a hardware timer (from the microcontroller) providing a tick every 1 millisecond. The scheduling is chosen to be non-preemptive; that is when a task is running while another task becomes ready to run, then the execution of the first task continues first then the other task is run.

Static Architecture

Initially, a simple CAN communication stack was implemented. Then, the protocol layer was added to it. Finally, the software architecture is as shown in figure 3.15:

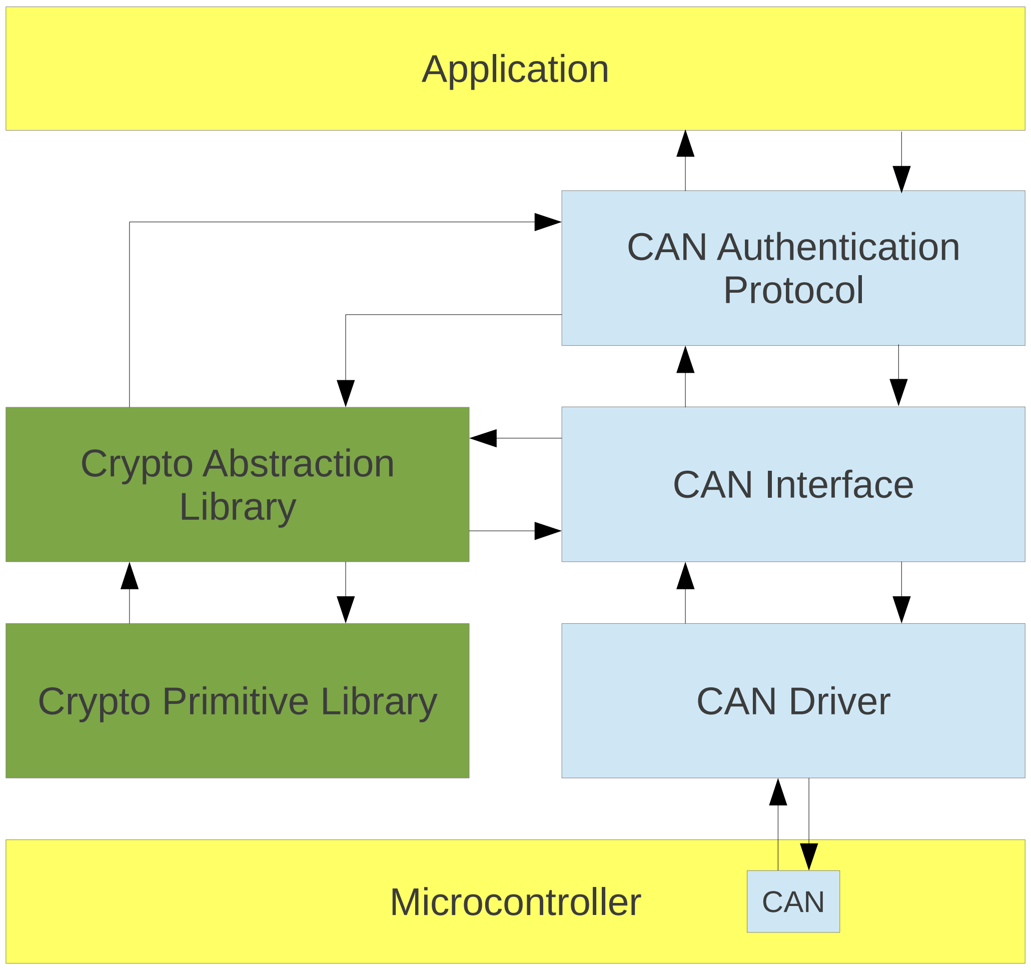


Figure 3.15: Software Architecture

As shown in the figure, the main components are:

- **CAN Driver (CanDrv)**

  The "CAN Driver" is the component which interfaces with CAN peripheral of the microcontroller. It configures the baudrate of CAN and configures different message buffers [46]. It provides APIs for setting and getting data to / from message buffers.

- **CAN Interface (CanIf)**

The “CAN Interface” is the component that provide APIs to the upper layers in order to transmit/receive different CAN messages. It uses “Crypto Abstraction Library” to encrypt/decrypt messages.

- **CAN Authentication Protocol (CanAP)**

  The “CAN Authentication Protocol” is the main component that implements the proposed authentication protocol. It uses “Crypto Abstraction Library” to generate/verify magic number chain.

- **Crypto Abstraction Library (CAL)**

  The “Crypto Abstraction Library” is the component responsible for abstracting the access to different cryptographic functions that are needed by the authentication protocol. It provides APIs responsible for performing various cryptographic functions such as encryption, decryption, one-way hash function transformation.

- **Crypto Primitive Library (CPL)**

  This component implements cryptographic primitive functions like encryption, decryption, hash functions, HMAC ... etc. It is actually, the PolarSSL library.

3.6.4 Measurements

A serial communication channel running at 38400 bps is used to make debugging easier. The channel utilizes the UART module of the microcontroller and is connected to a desktop computer from the other side. This channel is used by the software to print useful information to the desktop computer in order to trace the flow of software as well as printing measurements data.

The execution time of different functions is measure using a free running timer. This is done by configuring a timer inside the mircocontroller to run at frequency of 1 MHz. This provides a time measurement resolution of 1μs. In order to measure the time of the designated function only excluding the execution time of any other interrupt, we disable interrupts during the execution of the function.

Chapter 4

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 of case A 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 are taken into consideration:

- Hardware modifications
- Software and response time overhead
- CAN message set modifications
- Security Strength
- Robustness
- Scalability
- Maintainability

4.1 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 [15] that needed new CAN controller and CAN transceiver. It is also better than using Hardware Security Module (HSM) defined in EVITA project [10] 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.

The only hardware modifications that may be needed are those needed when either the additional CPU load or memory consumption is beyond the capacity of the used ECU.

4.2 Software and Response Time Overhead

The software overhead required for the authentication protocol can be classified into the following:

- CPU load and memory consumption needed for performing cryptographic calculations mainly as well as 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.

4.2.1 CPU Load Formula

The additional CPU load required for using the authentication protocol in a CAN node can be calculated as follows:

For each transmitted message, the protocol needs additional CPU load for encryption, HMAC and hash. Similarly for each received message, the protocol needs additional CPU load for decryption, HMAC and hash.

Hence, the overhead for each transmitted message i is:

$$\text{Overhead Tx}|_{Message_i} = (\text{Encryption Overhead } (O_e)\text{+HMAC and hash Overhead}(O_h))$$

Therefore, if the node transmits n_{Tx} messages; each is sent with periodicity P_i, then the total CPU load for all transmitted messages can be expressed as follows:

$$\text{CPU Load Tx}|_{Node} = \sum_{i=1}^{n_{Tx}} \frac{O_e + O_h}{P_i}$$

Since the overhead is constant for all messages, then:

$$\text{CPU Load Tx}|_{Node} = (O_e + O_h) \times \sum_{i=1}^{n_{Tx}} \frac{1}{P_i}$$

Define the "Tx Load Factor" as:

$$TxLF|_{Node} = \sum_{i=1}^{n_{Tx}} \frac{1}{P_i}$$

As a result, the CPU load for transmitted message by the node is:

$$\text{CPU Load Tx}|_{Node} = (O_e + O_h) \times \text{TxLF}|_{Node} \qquad (4.1)$$

Similarly, the CPU load required for n_{Rx} received messages can be expressed by the equation:

$$\text{CPU Load Rx}|_{Node} = (O_d + O_h) \times \text{RxLF}|_{Node} \qquad (4.2)$$

where O_d is the overhead due to decryption and RxLF is the "Rx Load Factor" which is defined as:

$$RxLF|_{Node} = \sum_{i=1}^{n_{Rx}} \frac{1}{P_i}$$

According to CAN message sets obtained from [47], we calculate the Tx and Rx load factors of 3 ECUs based on the periodicity of messages that each of them transmits/receives. The results is shown in the table below:

	ECU 1	ECU 2	ECU 3
Tx Load Factor	0.071	0.23	0.017
Rx Load Factor	0.491	0.216	0.139

Conclusion

As shown in equations 4.1 and 4.2, the additional CPU load that is required for the proposed authentication protocol is directly proportional to both the overhead and the load factor. As defined above, the load factor depends on the number of transmitted/received CAN messages and their periodicity. To avoid changes to the transmitted/received messages, the value of the load factor is assumed to be constant for each node. As a result, the additional CPU load that is required can be only optimized by optimizing the CPU load needed for the overhead. This can be done in two ways; optimizing encryption/decryption and optimizing HMAC.

4.2.2 Encryption/Decryption Results

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.

For "Session Key", each node uses a single session key for all messages that it transmits while it uses a separate session key for each group of messages that it receives from a given transmitting node. Therefore, the needed RAM for this operation is:

$$\text{RAM Consumption in bytes} = 10 + (10 \times \text{Number of transmitting nodes})$$

Where "Number of transmitting nodes" is the number of nodes that transmits messages to the node for which we are measuring the needed RAM.

Therefore,

$$\text{RAM Consumption in bytes} = 10 \times (1 + \text{Number of transmitting nodes}) \quad (4.3)$$

The formula above applies for "Extended Mode". For "Standard Mode", the number of encrypted/decrypted bytes is 8 instead of 10. Hence, it is required to cache 8 bytes only of the generated stream. Therefore, the formula takes the form:

$$\text{RAM Consumption in bytes} = 8 \times (1 + \text{Number of transmitting nodes}) \quad (4.4)$$

Encryption/Decryption Optimization

In general, the execution time of applying a stream cipher consists of two parts:

- Generation of a pseudo-random key stream
- Doing an XOR operation between the generated stream and the stream to be encrypted/decrypted.

Since the generated stream is the same for every encryption/decryption operation of the same key, then this byte stream can be cached in order to optimize the execution time. However, the cost of this optimization technique is that 10 bytes are needed from RAM in order to cache the generated pseudo-random stream.

The obtained result for doing XOR operation only took 4 μs.

HMAC and Hash Results

Regarding hashing, the execution time for SHA224 was 146 μs.

For HMAC, the execution time was initially long (before optimization). This step is needed to be performed at the receiver side for each received CAN message. According to the acceptance of how many messages that may be dropped, this time can be needed many times per message.

In order to have a uniform load at the sender side when refreshing the 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.

Using PolarSSL library (without optimization), the execution time in μs of HMAC using different hash functions varying the key size is as shown in table 4.1.

After HMAC optimization, the execution time in μs for the first run of HMAC is as shown in table 4.2.

Table 4.1: Execution time in μs of one HMAC operation of unoptimized code

Key Length / Hash Function	16 bits	32 bits	64 bits	80 bits	128 bits
MD5	150	152	154	155	158
SHA 1	286	288	290	291	294
SHA 224	543	544	546	546	549
SHA 256	543	544	546	547	550

Table 4.2: Execution time in μs of one HMAC operation of optimized code for first run

Key Length / Hash Function	16 bits	32 bits	64 bits	80 bits	128 bits
MD5	144	145	146	147	150
SHA 1	278	280	282	283	286
SHA 224	539	540	542	544	547
SHA 256	540	541	542	544	547

The execution times in μs for subsequent runs of HMAC has reduced times as shown in table 4.3.

Table 4.3: Execution time in μs of one HMAC operation of optimized code for subsequent runs

Key Length / Hash Function	16 bits	32 bits	64 bits	80 bits	128 bits
MD5	86	86	86	86	87
SHA 1	154	154	154	154	154
SHA 224	285	286	285	285	285
SHA 256	285	285	286	285	285

It is noticed that for the chosen key lengths, the length of the key does not affect much the execution time of HMAC. This is because for all key lengths that are below the input block size of the hash function, there is no much difference in calculations since the key is used directly without hashing. Changing the key length affects only the calculation of “ipad” and “opad”; which is done using a simple XOR operation.

HMAC Optimization

The initial results indicated that the time consumed for calculating HMAC is too long. As a result, we needed to optimize the implementation of HMAC. In order to do that, we first take an overview on the HMAC algorithm itself. Figure 4.1 describes the steps of the algorithm [34]:

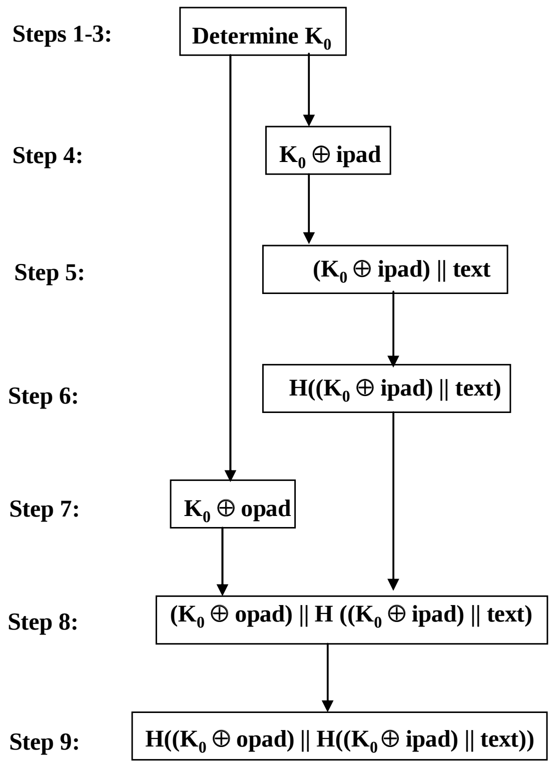

Steps 1-3:
Determine K_0
Step 4:
$K_0 \oplus$ ipad
Step 5:
$(K_0 \oplus \text{ipad}) \| \text{text}$
Step 6:
$H((K_0 \oplus \text{ipad}) \| \text{text})$
Step 7:
$K_0 \oplus$ opad
Step 8:
$(K_0 \oplus \text{opad}) \| H((K_0 \oplus \text{ipad}) \| \text{text})$
Step 9:
$H((K_0 \oplus \text{opad}) \| H((K_0 \oplus \text{ipad}) \| \text{text}))$

Figure 4.1: HMAC Algorithm

Define the following:

- K: The key to be used for HMAC.
- B: The input block size of the used hash function.
- L: The length of the output of the used hash function.
- H: The hash function that is used to calculate the HMAC.
- text: The input text to HMAC.
- ipad: The value 0x36 repeated B times.
- opad: The value 0x5C repeated B times.

• Steps 1 to 3: In these steps, the key is preprocessed so as to match the input block size of the used hash function.

  - If the length of $K = B$: set $K_0 = K$.
  - If the length of $K > B$: hash K to obtain an L byte string, then append $(B - L)$ zeros to create a B-byte string K_0
  - If the length of $K < B$: append zeros to the end of K to create a B-byte string K_0.

- Step 4: Exclusive-Or K_0 with ipad to produce a B-byte string.
- Step 5: Append the stream of data text to the string resulting from step 4.
- Step 6: Apply H to the stream generated in step 5.
- Step 7: Exclusive-Or K_0 with opad.
- Step 8: Append the result from step 6 to step 7.
- Step 9: Apply H to the result from step 8.

Measurements showed that the most consuming part is the hash function. Generally, a single HMAC calculation requires from 2 to 3 hash operations. In our case, the length of the key is less than the block size of the used hash functions. Therefore, a single HMAC calculation requires only 2 hash operations.

As indicated in [33], the time consumed in HMAC calculation can be optimized by doing the part of hash function on each of *ipad* and *opad* only once per used key.

4.3 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.

Typically, The message sets are defined by the vehicle manufacturer. For each manufacturer, there are some similarities in the message sets of each vehicle designed at the same time. Also, when designing a new vehicle, a part of the message sets is reused. As a result, to avoid re-designing the message sets from scratch, any high level protocol shall impose minimum modifications.

For our protocol, the “Extended Mode” uses the “Extended Identifier” field of the CAN message to transfer authentication data. Hence, it does not require any modifications to be done to the CAN message sets. For “Standard Mode”, two bytes of the CAN message are used to transfer authentication data. Therefore, there are no required modifications for CAN messages whose payload is less than or equal to 6 bytes. However, modifications are required for CAN messages whose payload is greater than 6 bytes. The messages have to be formatted so that the maximum payload of a message is 6 bytes.

Comparing our protocol to CANAuth, we find that for “Extended Mode” both protocols are good. However, CANAuth is superior to the “Standard Mode” of our proposal.

4.4 Security Strength

This section discusses the security strength of the proposed authentication protocol for different attack scenarios. The attack scenarios span different phases of the protocol. In our discussion, we recall the two cases of the threat model that we highlighted in the previous chapter:

- Case A: Existing ECU is compromised
- Case B: Adversary node connected to the bus

4.4.1 Channel Setup Attack

During “Channel Setup” phase, the sender sends both the “Session Key” , the “Channel Initial Magic Number” and the “HMAC Key” to each of its receivers. As mentioned earlier, these parameters are encrypted using a pre-shared key between the sender and each of its receivers. A successful attack can result in either deducing the pre-shared key itself or at least being able to send false parameters to a single receiver.

As calculated earlier in chapter 3, a brute-force attack that aims at deducing the 128-bit pre-shared keys needs 5.4×10^{30} years.

The other type of attack is either masquerading the sender or receiver.

Attacker masquerades a receiver

The attacker starts “Channel Setup” by sending channel setup request. This request can be either a new request or replaying a previously sent request by the original receiver. Since the request is encrypted using pre-shared key, then there is no difference between case A and case B because the pre-shared key is not known in both cases.

If the attacker replays a previously sent request, then it will have a correct checksum and verified by the sender. As a result, the sender continues the "Channel Setup" phase and send all data that is exchanged during this phase. For case A, the data sent is not useful to the attacker because it already knows it. For case B, the data is not useful because the attacker does not know either the pre-shared key or session key.

On the other hand, if the attacker tries to generate a new request, then it will put a random value for the checksum because it does not know the pre-shared key. The probability of success of this step is 2^{-15} because the length of the checksum is 16 bits. In case of success, the result will be the same as the replay attack mentioned in the previous paragraph.

The effect of such attack is that it loads the sender and thus performing DoS attack on it. Therefore, the sender has to limit the number of channel setup requests that it can handle per unit time and also limit it to the initialization of the ECU only. In order to allow the original receiver perform the "Channel Setup' phase successfully, then it shall not use the same nonce twice. However, this requires that both the sender and receiver store previously used values of nonce so as not to repeat them.

Attacker masquerades a sender

The receiver initiates the "Channel Setup" phase normally by sending a channel setup request including a nonce. For case A and case B, the attacker may succeed in replying to the receiver with the following probabilities:

- Probability of replying with a correct hash(nonce) = 2^{-31}
- Probability of replying with a correct session key 1 = 2^{-31}
- Probability of replying with a correct session key 2 = 2^{-31}
- Probability of replying with a correct session key 3 = 2^{-31}
- Probability of replying with a correct HMAC key for case A = 2^{-15}
- Probability of replying with a correct HMAC key for case B = 2^{-31}

The probabilities listed above are calculated based on the fact that each handshake message shall have a correct checksum (16 bits) and is authenticated using a 16 bits magic number. The probability of sending a correct "HAMC key" differs in case A than in case B because the data of this step is encrypted using the session key which is known to the attacker in case A; the only unknown is the magic number used for authentication.

Thus,

- The probability of successful attack in case A = $4 \times 2^{-31} + 2^{-15}$.
- The probability of successful attack in case B = 5×2^{-31}.

The effect of such attack is that the receiver will get wrong values for channel setup and hence will not continue the following protocol phases successfully. Therefore, it will request "Hard Synchronization" after that. it is worth noting that as long as the attacker does not know the pre-shared key, then even if it succeeds in this phase then it will not be able to continue the following phases because it does not know the plaintext of the sent data.

4.4.2 Message Setup Attack

In "Message Setup" phase, the sender broadcasts the initial "magic number" of each message to all of its receivers. The goal of the attack on this phase is to send wrong values for initial magic numbers to the receivers. As a result, all messages that are sent by the attacker would be authenticated by the receivers while all the messages that are sent by the original sender are discarded.

Since the messages exchanged during this phase are encrypted using the "Session Key" then for an attacker to make a successful attack, he has to know the value of the "Session Key". Also, since the messages are authenticated using the channel initial magic number, then the attacker has to know its value as well as the HMAC key so that he can try to deduce the channel magic number chain.

For case A, both the "Session Key" and "HMAC Key" are known to the attacker. However, generating the magic number chain in a reversed order needs large memory for storing all possible chains. The memory needed to store all chains for a given HMAC key is about 128 KB. Since the length of HMAC key is 16 bits, then the total memory needed to store all chains for all

HMAC keys is $2^{16} \times 128KB$ which is equal to 8 MB. This value is larger than available memories for microcontrollers used in the automotive field. Also, this type of attack needs the compromised ECU to have enough computational power to search in this amount of data. If the attacker succeeds in this attack, it will be able to use the wrong magic numbers for sending false data messages during the "Data Exchange" phase.

For case B, the attacker tries to send random values till it gets authenticated. The probability of success of this attack is 2^{-15} because the length of the magic number used for authentication is 16 bits. Even if the attacker succeeds in this attack, it will prevent the receiver from doing the correct message setup. However, it will not be able to use the attack to send wrong authenticated data.

4.4.3 Data Exchange Attack

In "Data Exchange" phase, the sender broadcasts the messages that it sends to all receivers. Each message is being authenticated at each receiver by verifying the encapsulated "magic number". This phase can be attacked by masquerading the sender and sending false data messages instead of it.

For case A, the attacker already knows the values of "Session Key" and "HMAC Key". As a result, it knows the current value of the magic number. Therefore, if it can deduce the magic number chain in a reverse order then it may be able to send false messages that get authenticated by the receivers.

The other method of attack is to send random value for the magic number. This method can be done in both case A and case B. The probability of such attack is 2^{-15}.

A successful attack on this phase results in sending wrong data message and dropping the original one. This is for one message only; the following messages are not affected.

4.4.4 Chain Refresh Attack

In the "Chain Refresh" phase, the sender broadcasts the new initial magic number of a certain message to all of its receivers. The goal of the attacker in this phase is to send a false initial magic number and hence send false data to the receivers instead of the original data. This attack can be performed in the same way as the attacks performed in the "Data Exchange" phase with two exceptions:

1. The impact of the attack is larger. The attack on "Data Exchange" message results in dropping a single data message. However, a successful attack on this phase give the attacker full control on the data channel and thus being able to send multiple false messages while all messages sent by the original sender fails to get authenticated.
2. The attack has to be performed in a specific period of time; the time where it is expected to send a chain refresh message. This time comes whenever a magic number chain is consumed.

4.4.5 Soft Synchronization Attack

In "Soft Synchronization" phase, the receiver requests from the sender the current values of magic numbers of all the messages that it sends. There are two types of attacks on this phase. The attacker may either masquerade the sender or the receiver.

Attacker masquerades a receiver

When the attacker masquerades the receiver, it sends false "Soft Synchronization" request to the sender. For case A, the attacker will receive the current values of magic numbers which it may already know. For case B, the attacker will not be able to decrypt the values of magic numbers. Therefore, the benefit of this type of attack is to perform DoS attack on the sender; making it busy serving many soft synchronization requests.

Attacker masquerades a sender

When the attacker masquerades the sender, it tries to reply to soft synchronization request received from a certain receiver.

For case A, the attacker may succeed in replying to the receiver with the following probabilities:

- Probability of replying with a correct hash(nonce) = 2^{-31}
- Probability of replying with a correct magic number = 2^{-15}

When such attack succeeds, the attacker will be able to send data instead of the original sender.

For case B, the attacker may succeed in replying to the receiver with the following probabilities:

- Probability of replying with a correct hash(nonce) = 2^{-31}
- Probability of replying with a correct magic number = 2^{-31}

When such attack succeeds, the attacker will not be able to send data instead of the original sender. However, it will prevent the receiver from synchronizing correctly with the sender.

4.4.6 Hard Synchronization Attack

The analysis of attacks on this phase is the same as the attack on both "Channel Setup" and "Message Setup".

4.5 Robustness

This section discusses the robustness of the proposed CAN authentication protocol in various scenarios.

4.5.1 Message Loss

The environment inside a vehicle is susceptible to high levels of electromagnetic interference. This concern was always taken into consideration when designing various automotive communication bus protocols. Similarly, this concern has to be also considered for any new protocol. The high levels of electromagnetic interference inside the vehicle environment can cause message losses. For the case of CAN bus, one or more CAN messages may be lost. ECUs should be able to recover from this incident.

Our protocol behaves in the case of CAN messages loss as follows. When a message or more is lost, the receiver cannot verify the magic number using a single one-way hash function transformation. As specified earlier, the protocol parameter δ specifies the maximum number of trials that a receiver may perform in order to verify the received magic number.

The example shown in figure 4.2 shows a loss of 3 consecutive CAN messages. The receiver receives the message with magic number m_i then 3 messages are lost and it receives after that the message with magic number m_{i+4}.

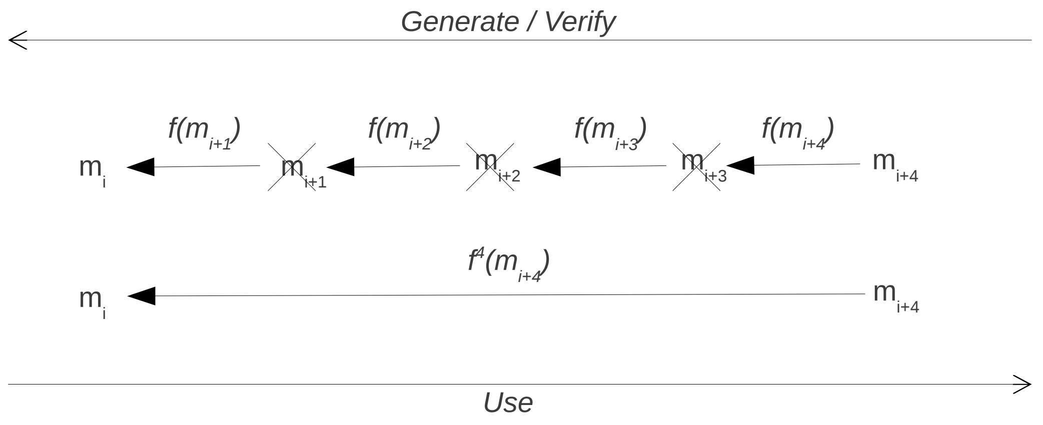


Figure 4.2: Loss of 3 messages

In this case, the receiver response depends on the value of the protocol parameter δ. There are two cases to consider:

1. $\delta \geq$ Number of lost messages $+ 1$
2. $\delta <$ Number of lost messages $+ 1$

If the value of the protocol parameter δ is greater than or equal to 4, then the receiver will do the following. The receiver applies the one way hash function to m_{i+4}, compares it to m_i, but finds that the 2 numbers are different. The receiver performs another transformation, but again the resultant value is not equal to m_i. The receiver continues that way till it makes the fourth transformation. The resultant number of this transformation is the same as m_i. As a result, the message is verified and the current magic number is updated with the value of m_{i+4}.

If the value of the protocol parameter δ is less than 4, then the receiver will do the following. The receiver applies the one way hash function to m_{i+4}, compares it to m_i, but finds that the 2 numbers are different. The receiver keeps performing another transformations (according to the value of δ), but all trials fail. As a result, the receiver confirms that it lost synchronization with the current value of the magic number and sends a "Soft Synchronization" request to the sender.

For the first case, if the lost messages are at the end of magic number chain, then even when the receiver performs δ transformations, it will not verify the magic number. In this case, the receiver confirms that it lost synchronization and it sends "Soft Synchronization" request (The same as in the case of $\delta < 4$).

It is clear that increasing the value of *delta* increases the protocol strength towards messages loss. However, this comes on the cost of increasing the CPU load. The needed CPU load for each trial can be calculated by the CPU load formula mentioned earlier in this chapter.

4.5.2 ECU Reset

Any ECU may reset suddenly to any reason. In this case, it shall be possible for the communication to be resumed successfully. Since any ECU has two roles; transmitter and receiver, then it will behave differently for each role.

As a receiver, after the ECU resets, it will wait for "Channel Setup" requests to be sent to it. "Channel Setup" requests are expected to be sent to it in the case of a new driving cycle. However, if the ECU resets during normal operation of the vehicle, such requests will not be received. After a certain timeout period when such requests are not received, it will start to send "Hard Synchronization" requests to each sender that it communicates with. After the success of the "Hard Synchronization" phase, it will be communicating normally with others ECUs.

On the other hand - the ECU as a transmitter - will start the normal phases of the protocol by performing "Initialization", "Channel Setup" and "Message Setup" before it starts "Data Exchange".

4.6 Scalability

In order to discuss the scalability, two parameters are studied:

1. Number of ECUs in the CAN network (N)

2. Number of exchanged messages (M)

We study the effect of changing each of the two studied parameters in each phase of the protocol.

4.6.1 Initialization

In the initialization phase, each sender creates the “Session Key”, the “HMAC Key”, “Handshaking Magic Number Chain”, “Channel Magic Number Chain” and magic number chains for each message that it sends.

Neither the generation of “Session Key” nor the generation of “HMAC Key” is affected by increasing N.

The time and memory needed for the generation of “Handshaking Magic Number Chain” increases linearly with N. However, it is not affected by M. This is because the sender generates a separate chain for each receiver.

The length of the “Channel Magic Number Chain” increases linearly with increasing M while it is not affected by increasing N.

The number of generated chains for each message increases linearly by increasing M while it is not affected by increasing N.

4.6.2 Channel Setup

The time consumed in “Channel Setup” phase increases linearly by increasing N because channel setup is performed for each pair of a sender and receiver. However, increasing M has no effect on this phase.

4.6.3 Message Setup

The time consumed in “Message Setup” phase increases linearly by increasing M. However, it is not affected by increasing M.

4.6.4 Data Exchange

The time consumed in the “Data Exchange” phase is not affected by either increases N or M. From this point of view, the proposed protocol is better than [18]; where a message authentication code (MAC) is added inside the payload of the message for each receiver.

4.6.5 Chain Refresh

The time consumed in a single “Chain Refresh” phase is not affected directly by either increasing N or M. However, the phase itself is performed for each message and hence increases linearly by increasing M.

4.6.6 Soft Synchronization

The time consumed in “Soft Synchronization” increases linearly by increasing M. There is no direct relation between increasing N and the time of each “Soft Synchronization” phase. However, increasing N increases the probability of having more than one receiver requesting a “Soft Synchronization” phase at the same time.

4.6.7 Hard Synchronization

The scalability of “Hard Synchronization” phase is the same as “Soft Synchronization” phase.

4.7 Maintainability

When it is required to replace an ECU that implements the authentication protocol, the pre-shared keys that are used for the “Channel Setup” phase have to be updated so that the new ECU communicates correctly with other existing ECUs. We propose any of the following different ways for updating the keys:

- Calibrate the values of pre-shared keys in the new ECU with the same values as of the old ECU. This requires that the values of keys are given to the owner of the car to store them in a safe place.
- Calibrate the values of pre-shared keys in existing ECUs according to the values programmed inside the new ECU. Strong security access is needed so that calibrating existing ECUs is done by a trusted person only.
- Calibrate the values of pre-shared keys in both the new ECU and other existing ECUs. This requires the same precautions described in the previous two methods.
- Do self calibration for the new ECU together with other existing ECUs. In other words, use a key generation scheme like Diffie-Hellman to generate the new values of pre-shared keys.

Selecting any of the methods described above is beyond the scope of the thesis. The most important thing is to make sure that this process is done in a secure way and at the same time enable vehicle owners to replace ECUs easily.

Chapter 5

Conclusion and Future Work

5.1 Future Work

5.1.1 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.

5.1.2 Enhance the maintainability of the protocol

The protocol assumes that each two communicating ECUs have a pre-shared key for communicating with each other. This introduces a limitation when any ECU needs to be replaced. As seen in the previous chapter, many solutions for this problem have been discussed. However, there is still a need to define a clear solution for this maintainability issue.

5.1.3 Reduce the execution time of one-way hash function

It is required to optimize the execution time of the used one-way hash function. This can be achieved either by code optimization or by modifying the algorithm itself. Both ways should be useful since we truncate the output of the HMAC function.

5.1.4 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 [28] which is already deployed in many cars. It offers better opportunity for authentication since its size can reach up to 254 bytes.

The other strong candidate is One Pair Ethernet [48]. 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.

5.2 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 protocols have been proposed before but their adoption was not easy because they either needed modifications in the physical layer of the CAN [15] or they added much overhead inside the CAN message [18] that reduced the size of the useful payload inside the message.

The main goal of the work presented in the thesis 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. FlexRay with a payload up to 254 bytes per frame is a good alternative from security point of view. Also, One Pair Ethernet - as an emerging method - looks more promising.

References

[1] R. N. Charette, “This car runs on code,” *IEEE Spectrum*, Feb 2009.

[2] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, and S. Savage, “Experimental security analysis of a modern automobile,” in *Security and Privacy (SP), 2010 IEEE Symposium on*, Oakland, CA, USA, May 2010, pp. 447–462.

[3] “Delivering online capabilities on the road,” Alcatel-Lucent, Tech. Rep., 2010.

[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] *Road vehicles – Vehicle identification number (VIN) – Content and structure*, ISO Std. 3779, 2009.

[6] M. Ghangurde, “Ford’s SYNC® and microsoft windows embedded automotive make digital lifestyle a reality on the road,” SAE Inetrnational, 2011.

[7] B. Niedermaier, S. Durach, L. Eckstein, and A. Keinath, “The new BMW iDrive applied processes and methods to assure high usability,” in *Digital Human Modeling*, ser. Lecture Notes in Computer Science, V. Duffy, Ed. Springer Berlin / Heidelberg, 2009, vol. 5620, pp. 443–452.

[8] R. Bose, J. Brakensiek, and K.-Y. Park, “Terminal mode transforming mobile devices into automotive application platforms,” in *Proceedings of the*

Second International Conference on Automotive User Interfaces and Interactive Vehicular Applications (AutomotiveUI 2010), Pittsburgh, Pennsylvania, USA, Nov 2010.

[9] *Specification of the radio data system RDS for VHF/FM sound broadcasting in the frequency range from 87.5 to 108.0 MHZ*, International Electrotechnical Commission Std. IEC 62 106, 1999.

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

[11] 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.

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

[13] 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.

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

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] ——, "Flexible multicast authentication for time-triggered embedded control network applications," in *Proc. of the International Conference on Dependable Systems and Networks (DSN 09)*, Lisbon, Portugal, Jun/Jul 2009, pp. 165–174.

[20] 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.

[21] 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

[22] 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.

[23] 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.

[24] 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

[25] 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.

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

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

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

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

[30] G.-N. Sung, C.-Y. Juan, and C.-C. Wang, “Bus guardian design for automobile networking ecu nodes compliant with flexray standards,” in *Consumer Electronics, 2008. ISCE 2008. IEEE International Symposium on*, Apr 2008, pp. 1 –4.

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

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

[33] W. Stallings, *Cryptography and Network Security*, 4th ed. Prentice Hall, 2006.

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

[35] R. Rivest, “The MD5 Message-Digest Algorithm,” RFC 1321 (Informational), Internet Engineering Task Force, Apr 1992, updated by RFC 6151. [Online]. Available: http://www.ietf.org/rfc/rfc1321.txt

[36] T. Xie and D. Feng. (2009) How to find weak input differences for md5 collision attacks. Cryptology ePrint Archive, Report 2009/223. [Online]. Available: http://eprint.iacr.org/2009/223

[37] Y. Sasaki and K. Aoki, “Finding preimages in full MD5 faster than exhaustive search,” in *Advances in Cryptology - EUROCRYPT 2009*, ser. Lecture Notes in Computer Science, A. Joux, Ed. Springer Berlin / Heidelberg, 2009, vol. 5479, pp. 134–152.

[38] M. Mao, S. Chen, and J. Xu, "Construction of the initial structure for preimage attack of MD5," *Computational Intelligence and Security, International Conference on*, vol. 1, pp. 442–445, 2009.

[39] *Secure Hash Standard (SHS)*, National Institute of Standards and Technology Std. FIPS PUB 180-3, 2008.

[40] "Announcing request for candidate algorithm nominations for a new cryptographic hash algorithm SHA3 family," National Institute of Standards and Technology, 2007.

[41] Cryptographic hash algorithm competition. National Institute of Standards and Technology. Last accessed August 2012. [Online]. Available: http://csrc.nist.gov/groups/ST/hash/sha-3/index.html

[42] eBASH - ECRYPT benchmarking of all submitted hashes. European Network of Excellence in Cryptology II. Last accessed August 2012. [Online]. Available: http://bench.cr.yp.to/ebash.html

[43] XBX - eXternal Benchmarking eXtension. European Network of Excellence in Cryptology II. Last accessed August 2012. [Online]. Available: http://xbx.das-labor.org/trac

[44] SUPERCOP. European Network of Excellence in Cryptology II. Last accessed August 2012. [Online]. Available: http://bench.cr.yp.to/supercop.html

[45] Light-weight, modular cryptographic and ssl/tls library in c - polarssl. Offspark. Last accessed August 2012. [Online]. Available: http://polarssl.org/

[46] *MPC5604B/C Microcontroller Reference Manual*, Freescale Semiconductor, May 2011, rev. 8.

[47] "Confidential information obtained from OEMs."

[48] 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.

الملخص

لقد ساعد تطور الالكترونيات خلال القرن الماضى فى تغيير طبيعة السيارات بشكل كبير. ففى الوقت الحاضر يتم التحكم فى مختلف الأنظمة داخل السيارة الواحدة عن طريق العشرات من وحدات التحكم الالكترونية. تلك الوحدات تتصل ببعضها البعض عن طريق شبكات اتصال بينها بعض البوابات. و قد تم ا سخدام العديد من الأنظمة القيا سية فى الاتصال داخل هذه الشبكات و بين بعضها البعض. و يعد نظام شبكة المتحكم من أو سع الأنظمة انتشارا فى وقتنا هذا. و بصفة عامة فقد كان تصميم هذه الشبكات دوما ما يهتم بأمور السلامة و الاعتما دية. و لم يكن هناك اهتمام بشأن تأمين هذه الشبكات. و يرجع السبب فى ذلك لعدم وجود دليل واضح على امكانية تعرض هذه الشبكات للاختراق. و لكن مؤخرا تم إجراء بعض التجارب العملية التى أظهرت تأثير بعض الهجمات على مختلف الأنظمة داخل السيارة. و قد شملت تلك التجارب وحدات التحكم المختصة بالتحكم فى المحرك و المكابح و الإضاءة و تكييف الهواء و جسم السيارة. و التالى بات من الواضح أنه بإمكان الشخص المخترق أن يسيطر على السيارة و بالتالى يصيب الركاب بالضرر. و هذا من شأنه يمثل خطراً حقيقياً لكل السيارات سواء المباعة أو التى ما زالت قيد التصميم.

و قد تم تصنيف الثغرات الأمنية التى تساعد على تنفيذ مثل هذه الهجمات إلى نوعين. النوع الأول هو الثغرات الناتجة عن ا ستخدام أنظمة الاتصال القيا سية. أما النوع الثانى فيشمل الثغرات الناتجة عن عدم الالتزام بمعايير التأمين. و قد دفعنا هذا إلى أن ندرس فى هذه الر سالة الطرق المختلفة لتأمين الشبكات الداخلية للسيارات. و تشمل الطرق المطروحة عدة مستويات من التأمين بدءاً من المنع مروراً بالحماية بالإضافة إلى الاكتشاف. و تقوم الر سالة بالتركيز على تأمين شبكة المتحكم من أى ا ختراق قد تتعرض له. و تعتمد طريقة التأمين المقترحة على ا ستخدام بروتوكول للمصا دقة على مصدر الر سائل المتداولة داخل شبكة المتحكم. و قد تم تحقيق ذلك عن طريق تصميم بروتوكول جديد للمصا دقة. و قد تم تنفيذ البروتوكول على متحكم دقيق من الفئة المستخدمة فى السيارات. و يتسم البروتوكول بأنه بسيط و يمكن تطبيقه عملياً مما يسهل من عملية ا ستخدامه مبا شرة فى صناعة السيارات. و لا يتطلب البروتوكول أى تعديلات فى المكونات المستخدمة و بالتالى يمكن ا سخدامه فى السيارات التى تم بيعها بالفعل.

مهنـــدس:	أحمد حازم جمال يوسف	ضع صورتك هنا
تاريخ الميلاد:	١٩٨٣/١٠/٠٣	
الجنسية:	مصرى	
تاريخ التسجيل:	٢٠٠٧/١٠/٠١	
تاريخ المنح:	/ /	
القسم:	الالكترونيات و الاتصالات الكهربية	
الدرجة:	ماجستير	

المشرفون:

ا.م.د. حسام على حسن فهمى

الممتحنون:

أ.د. أيمن محمد محمد حسن وهبة (أستاذ بكلية الهندسة–جامعة عين شمس)

أ.د. مجدى سعيد السودانى

أ.م.د. حسام على حسن فهمى

عنوان الرسالة:

طرق تأمين الشبكات الداخلية للسيارات

الكلمات الدالة:

سيارات، تأمين، مصادقة ، شبكة المتحكم ، حماية

ملخــص الرسالة:

كان تصميم الشبكات الداخلية للسيارات دوما ما يهتم بأمور السلامة و الاعتمادية .و لم يكن هناك اهتمام بشأن تأمين هذه الشبكات .و قد أظهرت التجارب العملية مؤخرا إمكانية تعرض تلك الشبكات للاختراق .و تقوم الرسالة بالتركيز على تأمين شبكة المتحكم من أى اختراق قد تتعرض له .و تعتمد طريقة التأمين المقترحة على استخدام بروتوكول للمصادقة على مصدر الرسائل المتداولة داخل شبكة المتحكم .و قد تحقق ذلك عن طريق تصميم بروتوكول جديد.

طرق تأمين الشبكات الداخلية للسيارات

إعداد

أحمد حازم جمال يوسف

رسالة مقدمة إلى كلية الهندسة، جامعة القاهرة
كجزء من متطلبات الحصول على درجة الماجستير
فى هندسة الالكترونيات و الاتصالات الكهربية

يعتمد من لجنة الممتحنين:

أ.م. د/ حسام على حسن فهمى المشرف الرئيسى

أ. د. / مجدى سعيد السودانى

أ. د. / أيمن محمد محمد حسن وهبة

كلية الهندسة ، جامعة القاهرة
الجيزة ، جمهورية مصر العربية
٢٠١٣

طرق تأمين الشبكات الداخلية للسيارات

إعداد

أحمد حازم جمال يوسف

رسالة مقدمة إلى كلية الهندسة، جامعة القاهرة
كجزء من متطلبات الحصول على درجة الماجستير
فى هندسة الالكترونيات و الاتصالات الكهربية

تحت إشراف

أ.م. د/ حسام على حسن فهمى

أستاذ مساعد
قسم الالكترونيات و الاتصالات الكهربية
كلية الهندسة — جامعة القاهرة

كلية الهندسة ، جامعة القاهرة
الجيزة ، جمهورية مصر العربية
٢٠١٣

Cairo University

طرق تأمين الشبكات الداخلية للسيارات

إعداد

أحمد حازم جمال يوسف

رسالة مقدمة إلى كلية الهندسة، جامعة القاهرة
كجزء من متطلبات الحصول على درجة الماجستير
فى هندسة الالكترونيات و الاتصالات الكهربية

كلية الهندسة ، جامعة القاهرة

الجيزة ، جمهورية مصر العربية

٢٠١٣