석사 학위논문
Master's Thesis

CAN에서 메시지 순서 기반의 스푸핑 공격 방어 및 임시 ID를 이용한 DoS 공격 방어 기법

A Countermeasure against Spoofing and DoS Attacks based on Message Sequence and Temporary ID in CAN

안 수 현 (安 洙 鉉 Ahn, Soo-Hyun)
정보보호대학원
Graduate School of Information Security

KAIST

2016

CAN에서 메시지 순서 기반의 스푸핑 공격 방어 및 임시 ID를 이용한 DoS 공격 방어 기법

A Countermeasure against Spoofing and DoS Attacks based on Message Sequence and Temporary ID in CAN

A Countermeasure against Spoofing and DoS Attacks based on Message Sequence and Temporary ID in CAN

Advisor : Professor Kim, Kwang Jo

by

Ahn, Soo-Hyun

Graduate School of Information Security

KAIST

A thesis submitted to the faculty of KAIST in partial fulfillment of the requirements for the degree of Master of Science in the Graduate School of Information Security . The study was conducted in accordance with Code of Research Ethics[1].

2015. 12. 02.

Approved by

Professor Kim, Kwang Jo

[Advisor]

---

[1]Declaration of Ethical Conduct in Research: I, as a graduate student of KAIST, hereby declare that I have not committed any acts that may damage the credibility of my research. These include, but are not limited to: falsification, thesis written by someone else, distortion of research findings or plagiarism. I affirm that my thesis contains honest conclusions based on my own careful research under the guidance of my thesis advisor.

CAN에서 메시지 순서 기반의 스푸핑 공격 방어 및 임시 ID를 이용한 DoS 공격 방어 기법

안 수 현

위 논문은 한국과학기술원 석사학위논문으로
학위논문심사위원회에서 심사 통과하였음.

2015년 12월 02일

심사위원장 김 광 조 (인)

심사위원 하 정 석 (인)

심사위원 신 승 원 (인)

MIS
20143390

안 수 현. Ahn, Soo-Hyun. A Countermeasure against Spoofing and DoS Attacks based on Message Sequence and Temporary ID in CAN. CAN에서 메시지 순서 기반의 스푸핑 공격 방어 및 임시 ID를 이용한 DoS 공격 방어 기법. Graduate School of Information Security . 2016. 37p. Advisor Prof. Kim, Kwang Jo. Text in English.

ABSTRACT

The development of Information and Communications Technologies (ICT) has affected various fields such as big data, mobile, wearable, and so on. In addition, the automotive field has been affected by ICT, and Electronic Control Units (ECU) have been introduced to control vehicles efficiently. A network for communication between ECUs was necessary, because each ECU cannot operate alone, necessitating data exchange. As a result, vehicle network protocols have been introduced such as Controller Area Network (CAN), Local Interconnect Network (LIN), and FlexRay. Due to this, ECUs can efficiently transfer data to other ECUs making vehicle control more efficient.

Among these, CAN is a standard vehicle network protocol, and almost all vehicles use CAN as their vehicle network protocol. Although CAN has been widely used, its security is very vulnerable, because of its characteristics such as the broadcast environment and arbitration process in CAN. By using vulnerable characteristics, spoofing and denial of service (DoS) attacks can be easily performed in CAN. To solve this vulnerability of CAN, many ideas have been suggested such as intrusion detection systems (IDS), hashed message authentication codes, AES, which is a type of cryptography algorithm, and so on. However, the suggested security solutions in CAN have some problems such as the increase of traffic, effect of existing systems, adoption costs, etc. In addition, because some ideas were not properly verified due to the characteristics of vehicles, these ideas cannot guarantee its efficiency or effectiveness.

In this paper, a security gateway that modifies the existing gateway in CAN is suggested for its improved defense against spoofing and DoS attack. In case of spoofing attack, it defends using a sequence of messages based on the driver's behavior. By making a table that stores a sequence of messages based on the driver's behavior, spoofing attacks can be detected and whether a message is an attack can be determined through a verification process using SipHash. Furthermore, a temporary ID using a seed and SipHash can be used to defend against DoS attacks.

To verify our proposed idea, OMNeT++, which is a network simulator, is used. The suggested idea shows a high detection rate and low traffic increase. In addition, in the case of a DoS attack, the suggested idea shows that a DoS attack has no effect by analyzing the frame drop rate.

Keywords: CAN, IDS, Security gateway, spoofing attack, DoS attack.

Contents

List of Tables

List of Figures

Chapter 1. Introduction

1.1 Overview

The development of ICT has affected various fields. Among these, automotive field has also been affected due to combining ICT. A vehicle that is considered as mechanical changed all of parts in vehicle to electronic. Introduction of ECU that controls engine, break, and transmission in vehicle much contributed to digitalization of vehicle. Through ECU, a control of vehicle can be easily convenience and controlled outside of vehicle through short communication such as smartphone, table pc, WiFi, and bluetooth and so on. As you can see figure 1.1, ECU occupies all of parts in vehicle. Now, ECU in vehicle is up to 70 and importance of ECU will increase as developing vehicle. There will be 100 ECUs in vehicle. Although control of vehicle was convenience due to ECU, vehicle network protocol was needed to communicate with other ECUs. To solve this problem, vehicle network protocol which are CAN, LIN, FlexRay was developed. Among these, CAN is a typical vehicle network protocol and used in almost vehicles as a standard of network protocol. As introducing CAN, ECUs could efficiently communicate with each other and the control of vehicle was more convenience.

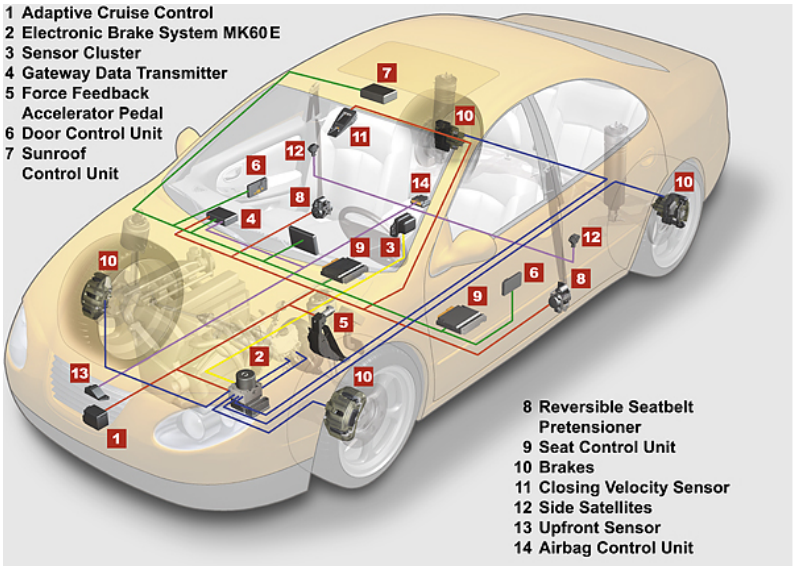

Figure 1.1: ECUs in vehicle

Although the introduction of ECU made vehicle control more efficient and convenient, security

research in CAN was rarely studied and many problems occurred due to a low awareness of vehicle security issues. Vehicle security research has gained importance due to recent hacking incidents involving GM and Chrysler vehicles. Because CAN has certain vulnerable characteristics, it is easy to hack vehicles employing such a system. For example, an ECU in the same network can receive messages without restriction, because CAN is a broadcasting environment. In such a situation, if a malicious attacker can seize control of or install an ECU into a CAN due to this characteristic, the attacker can learn about internal information and send fake messages by manipulating original messages. In addition, an attacker could make controlling a hacked vehicle difficult by making a DoS attack upon the CAN arbitration process. These attempts proved through many recent research [1][2][3][4] [5] [6] [7] and countermeasures about security in vehicle was important. To solve this problem, various research [8] [9] [10] studies have been undertaken. However, because it is difficult to build an experimental environment due to the high cost involved and accessibility issues, some ideas remain unproven or have inherent problems such as the volume of traffic, adoption costs, etc

Therefore, we propose an idea that can solve the problems of other ideas such as traffic and verification. By introducing a security gateway that modifies the existing gateway that CAN uses, we want to defend against spoofing and DoS attacks. Furthermore, by proposing an experiment result using OMNeT++, the proposed idea can effectively defend against spoofing and DoS attacks and demonstrate effectiveness by showing only a low increase of traffic.

1.2 Organization

The rest of this paper is organized as follows: Chapter 2 describes the background basic information about CAN, vulnerabilities in CAN, and related works about defense against attacks in CAN. In chapter 3, some assumptions are defined and ideas for systems that can defend against spoofing and DoS attacks through a security gateway are proposed. OMNeT++, which is used in experiments and experimental environments are also described in this chapter 4. Chapter 5 details the results and discusses the limitations. Finally, the conclusion and future work are discussed in Chapter 7.

Chapter 2. Background

2.1 CAN

CAN, which is a standard vehicle network, was proposed by Robert Bosch, which is a German company, at a 1982 conference of the Society of Automotive Engineers (SAE). Robert Bosch started developing new vehicle network protocols in 1983, because there was no network protocol that satisfied the demands of vehicle company engineers in the early 1980s. Mercedes Benz Engineers participated in the development stage. Bosch released CAN 2.0, which is a current specification, in 1991 and submitted it to the International Organization for Standardization (ISO). "ISO 11989" was released as a standard in 1991 and the maximum speed of the physical layer was defined as up to 1 Mbps. In addition, when an amendment was submitted to ISO, ISO 11989 was extended and a CAN that had a 29 bit ID field was introduced called "CAN 2.0B." However, because the released CAN specifications had some problems and imperfect parts, it caused a misunderstanding. To solve this, Bosch offered a CAN reference model and performed suitability tests on CAN controller chips. Benz released a vehicle that was equipped with CAN in 1992. The control of an engine system was performed using CAN at an initial step and the body system used CAN at the next step. After that, these two systems were connected to a gateway. European vehicle companies chose CAN as a vehicle network protocol and most vehicles used CAN, increasing vehicles' dependence on CAN. Figure 2.1 shows the share of the market that uses CAN. As you can see, CAN is not only used in the automotive field, but also medical and industrial fields, although usage of CAN is focused on the automotive field.

2.1.1 Characteristics

2.1.2 CAN Frame

The unit of data used in CAN is a frame. The CAN frame consists of four frames, which are data frame, remote frame, error frame, and overload frame. Among these, the data frame is important. Therefore, the data frame is described in detail and the others are only briefly explained.

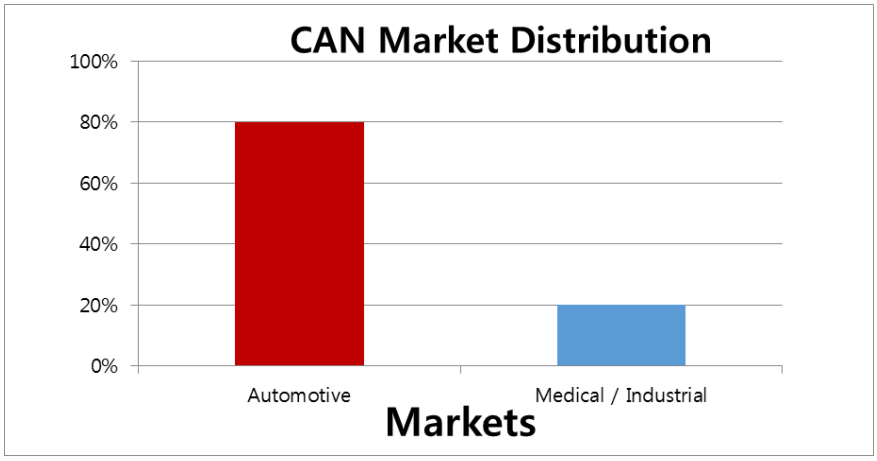

Figure 2.1: CAN market distribution

1) Data frame

- The data frame is used to send and receive data. The data frame consists of seven fields. Figure 2.2 shows the entire structure of the data frame.

- Start of Frame (SOF): This is the start of a frame.
- Arbitration Field (ID): This is used to identify messages and designate the order of priority. There are two formats, one is a standard format (CAN 2.0A), which uses an 11-bit ID field, the other is an extended format that uses a 29-bit ID field.

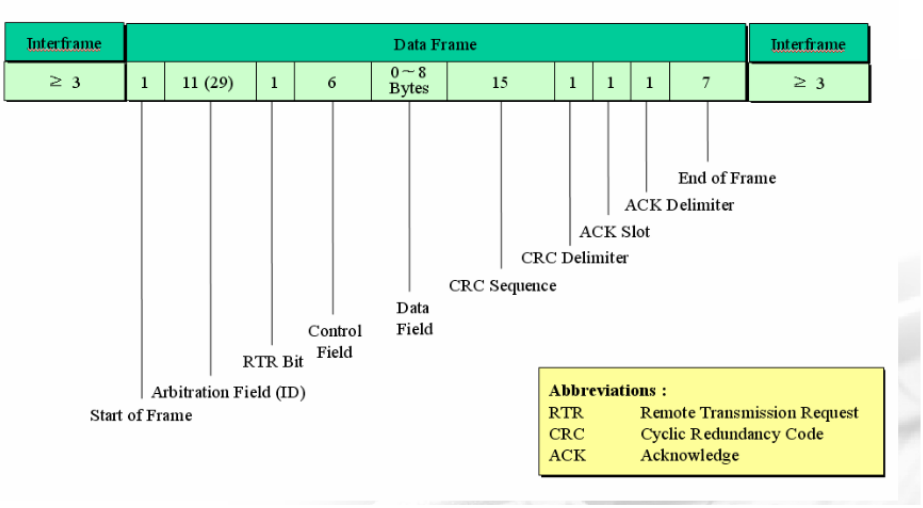

Figure 2.2: Structure of data frame

- Remote Transmission Request (RTR): This is used to distinguish the data frame from the remote frame. When the RTR bit is 0, the frame is a data frame. When the RTR bit is 1, the frame

becomes a remote frame.

- Control Field: This consists of a Data Length Code (DLC) field that shows the byte length and reserved bits, which are RB1 and RB2.
- Data Field: This is a field that sends data. It can have length of zero–eight bytes.
- CRC Field (CRC Sequence + CRC Delimiter): This consists of 15 bits of overlap check code and a delimiter bit. The CRC field is used to detect errors.
- ACK Field (ACK Slot + ACK Delimiter): All CAN controllers that accurately receive messages send an ACK bit at the end of messages. A sending node in bus checks whether the ACK bit exists in the bus and if there is no ACK bit, a node retries its messages.
- End of Frame (EOF): This represents the end of the frame.
- Interframe Field: This is positioned between frames. It allows frames to be distinguished.

2) Remote frame

- When one node in bus requires a data frame, this is used. At this time, The ID of the data frame and the ID of the remote frame are the same. The interframe space is placed between the data fra me and the remote frame to distinguish two frames. Figure 2.3 represents the structure of the remote frame. The difference between the data frame and the remote frame is that the RTR bit is 1 and there is no data field.

3) Error frame

- This is used to notify of an error in the bus. It consists of error flags and an error delimiter. The re are two error flags: One is the active error flag, the other is the passive error flag, and which fla g is received is determined according to the state of the node. The active error flag is six consecut ive dominant bits (0 of 6 bits) and the passive error flag is six consecutive recessive bits (1 of 6 bits). The error delimiter consists of eight consecutive recessive bits (1 of 8 bits).

4) Overload frame

- This is used to introduce delays between consecutive data frames or remote frames. The overload

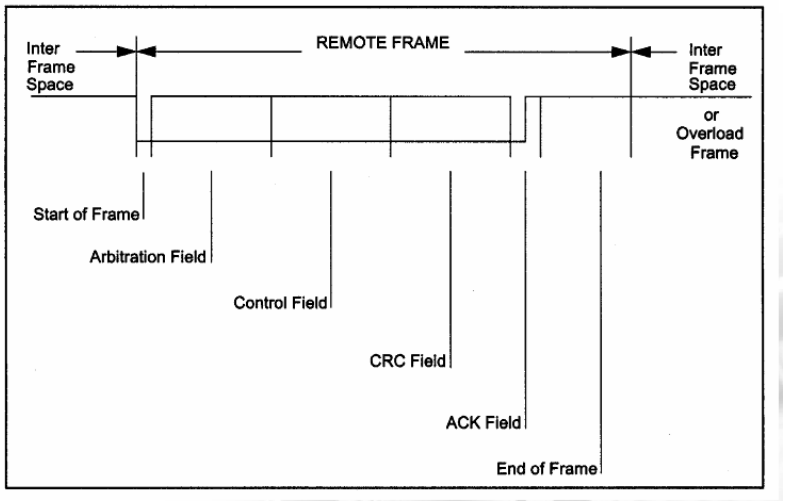

Figure 2.3: Structure of remote frame

frame consists of 15 bits and has an overload flag field and an overload delimiter field. Below are two cases in which the overload frame is sent to the bus.

- The internal state of the receiving node requires a delay in the next data frame or the remote frame.
- The dominant bit is detected in the inter-frame space.

2.2 Vulnerabilities in CAN

We examine characteristics and frames of the CAN in 2.1, as there are vulnerabilities in CAN caused by these characteristics. Below are the vulnerabilities of CAN and some possible attacks that use these vulnerabilities.

2.2.1 Broadcasting Nature

CAN uses broadcast, which means that all nodes in CAN receive messages without restriction. Therefore, it is easy to use a sniffing attack. Basically, a node can receive all messages from the bus, but does not handle messages that are sent to itself. However, if the attacker maliciously installs an ECU or seizes control of the ECU in CAN, attacker can then monitor messages in the CAN bus. In addition, using the On-Board Diagnostics-II (OBD-II), which is used for diagnosing the vehicle, it is easy to get messages from the CAN due to the broadcasting environment. An attacker can determine the meaning

of messages by collecting and analyzing such messages. Figure 2.4 shows a sniffing attack in the CAN. If the attacker installs a malicious ECU in the CAN bus, he can receive messages without restriction. Furthermore, it becomes possible to enact a spoofing attack. Because there is no sender information, nodes in CAN cannot know whether messages are from valid nodes. Therefore, after an attacker collects messages using sniffing and analyzes them, he can send malicious messages that include the ID used in the CAN bus. Because nodes cannot verify whether messages came from valid nodes, this attack will succeed.

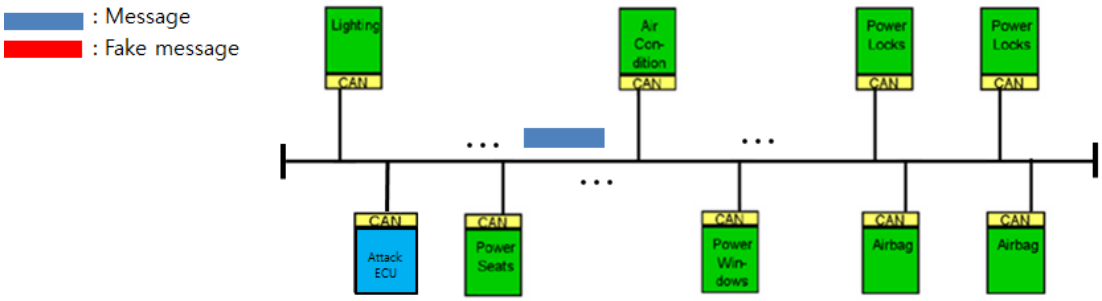

Figure 2.4: Sniffing attack in CAN

Figure 2.5 shows a spoofing attack in CAN. It shows how the attacker sends modified messages to the CAN bus through a malicious ECU that has been installed by the attacker. It is easy to use sniffing and spoofing attacks in such a broadcasting environment.

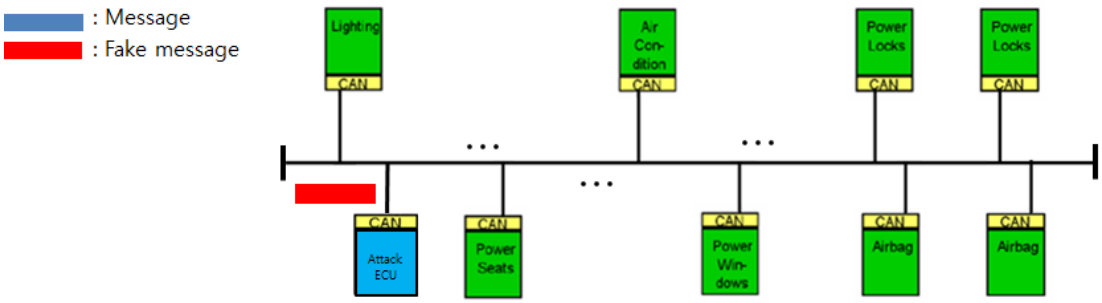

Figure 2.5: Spoofing attack in CAN

2.2.2 DoS

Next, CAN offers an autonomous arbitration process for avoiding collisions between messages when sending a message. Currently, the arbitration process is performed using an arbitration field in the data frame and two levels. As mentioned above, there are two levels (dominant and recessive), and the dominant(0) has a higher priority than the recessive(1). Due to this characteristic, a low ID has high

priority in the CAN. Figure 2.6 is an example of the arbitration process in CAN.

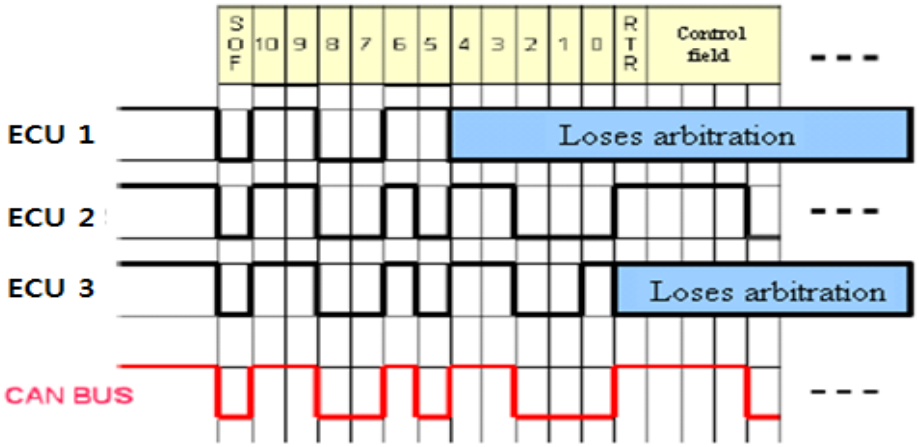

Figure 2.6: Arbitration process in CAN

There are three ECUs; once three ECUs start to send messages, each ECU monitors the variation of the arbitration field in the bus. Because ECU 1 is in the recessive state at the fifth and ECUs 2 and 3 are in the dominant state, ECU 1 loses its arbitration. Next, ECU 3 is in the recessive state at the zeroth and ECU 2 is in the dominant state. Therefore, ECU 3 loses priority and ECU 2 can finally send a message to the bus.

Using this characteristic, the attacker finds what ID has the highest priority in the CAN and creates messages that have the highest priority in the CAN. Then the attacker continuously sends such messages to the CAN bus. Because the attacker's messages have higher priority than the other nodes in the CAN, these nodes cannot send messages to the CAN bus. As a result, the vehicle cannot be operated, because messages cannot be sent. Figure 2.7 shows one such example.

As you can see, the attacker's messages and normal ECU's messages are together in the CAN bus. Because the attacker's messages have higher priority, normal ECU messages cannot be transferred due to the arbitration process in the CAN. This is termed a DoS attack, in which the attacker continuously sends fake messages to the CAN bus.

2.2.3 No Authentication

Finally, the biggest vulnerability in CAN is that there is no message authentication. Basically, the ID in the CAN frame is used for the transmission and arbitration processes. The problem that arises is that this ID does not represent the sender. Because this ID is used for whether messages are received at the receiver, nodes in the CAN cannot know where messages in the CAN originated. Therefore, even

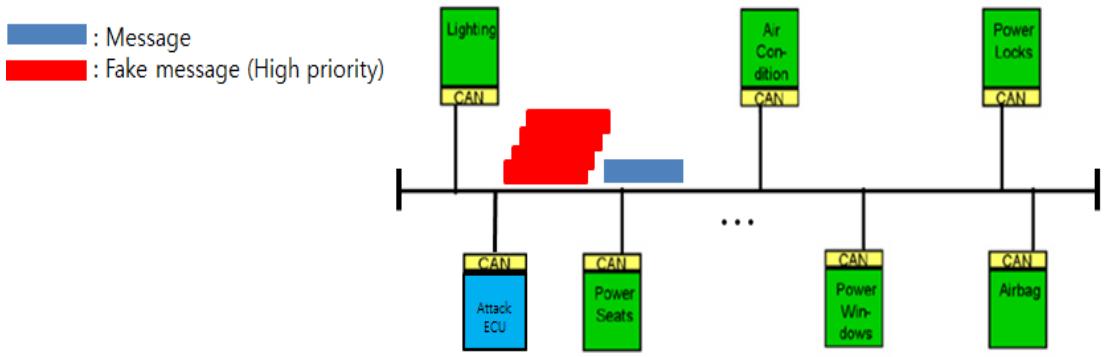

Figure 2.7: DoS attack in CAN

though an attacker sends fake messages that use message IDs obtained through sniffing, a receiver does not doubt the authenticity of any messages, because the attacker's message ID is valid. Thus, spoofing and DoS attacks are easily performed.

2.3 Related work

Many ideas have been suggested to defend against such security threats in the CAN. Among these, representatively ideas such as Intrusion Detection System (IDS), broadcast environment, Message Authentication code (MAC), and cryptography are typical methods for defending against security threats in the CAN. Below is an explanation of IDS, broadcast environment, MAC, cryptography, and driver behavior based.

2.3.1 IDS Based Methods

The IDS method has been suggested to detect spoofing and DoS attacks in the CAN. Two methods have been proposed that are similar to the existing IDS that is used in the network. One method uses the signature technique, the other uses the anomaly technique. In the case of the signature technique, the method using transfer characteristics in CAN is typical, unlike the existing method that examines the content of messages. Representatively, messages in the CAN are transferred periodically. Thus, a method using the periodicity of messages has been proposed. Some research [18][19][21][22] detected attacks using the periodicity of messages. In addition, a detection technique that uses a white-list has been suggested. The white-list technique[15] makes a table that stores the messages that are used in the CAN, detecting attacks by checking each ID in the table. This method is used in each ECU or gateway

in the CAN bus. Next, by combining the proposed methods[16][21][22] mentioned above, a method that checks the periodicity of messages, the ID, and so on has been suggested. Finally, there is a method[23] that has a monitor mode and a verification mode; messages are monitored using IDs in monitor mode. If there are suspect messages, the mode is changed to verification mode, and verification is processed using MAC. Although methods using a signature technique have been suggested, there is a problem that the existing ECU must be greatly modified. This lays a burden on the ECU and that becomes a problem, because lightweight is an essential factor in defending against attacks in the CAN. In addition, there is a verification problem. Some ideas could not be implemented and demonstrate efficiency or effectiveness through experimentation. Therefore, ideas cannot be proved good or bad.

Next, IDS using an anomaly technique has been suggested, but there are a few methods of achieving this. The method[19] set specific criteria such as a standard for messages, domain, etc., and if messages deviate from the predefined standard, it detects an attack. There is also the method[27] that uses message entropy. However, method using anomaly techniques are suggested less often because it is difficult to define anomalous behavior. In a vehicle environment, the driver's behavior has a huge effect on many things, and a driver may sometimes behave in an unexpected manner. Furthermore, the anomaly technique basically uses machine learning, but it is difficult to develop this and apply it to the ECU due to the ECU's resource constraint.

2.3.2 Broadcast Environment Based Methods

The broadcast environment based is a method[17] using the characteristic of broadcasting. It checks whether a message has been made by itself using the message ID. If messages have not originated within itself, it notifies the other ECUs in the CAN by sending error frames. In other words, if the attack ECU creates messages using a specific ID, the ECU that has the corresponding ID can know that messages that are broadcast in the CAN bus are invalid by checking the messages. When suspect messages are detected, the corresponding ECU can defend the system by notifying other ECUs that there is an attack message in the CAN bus by transmitting error frames. This method can detect spoofing attacks, but it cannot defend against a DoS attack. In addition, it has not been verified through experimentation, so its efficiency remains unproven.

2.3.3 MAC Based Methods

Using MAC to check message integrity has been applied to CAN. Various ideas[11][13][14][24][27][28] that use MAC have been proposed, and many methods use an existing hash-based MAC algorithm, CBC-MAC, or autonomously developed MAC algorithm. However, they encounter similar problems, in that they increase traffic. In case of the ideas suggested above, any length larger than 64 bits, which is the maximum length of data in CAN, is used for defending against attack. Thus, additional messages are created for the authentication of integrity. In the worst case, there will be four additional messages about one message. Due to the additional messages, the traffic in CAN will increase, which has an effect in real time, which is considered an important characteristic in a vehicle. In other words, there will be the problem that real time transmission cannot be guaranteed, creating a huge hazard.

2.3.4 Cryptography Based Methods

Methods using cryptography algorithms have also been suggested. A method[20] using AES, which is a type of cryptography algorithm, was suggested. In addition, there is method[12] that shares a pair-wise secret key beforehand between ECUs which is used to encrypt messages. In this situation, forgery prevention and the authentication of messages are both performed by the MAC. The ideas suggested above ensure confidentiality and integrity that CAN does not offer, but this approach has weaknesses such as padding value and increasing traffic. The length of messages in CAN varies because each manufacturer assigns the meaning of data differently. This means that data length can be less than 64 bits; thus, padding values must be arbitrarily added to data when encrypting messages using AES. When adding padding values to data, the padding value must be different. If the padding value is always the same, an attacker can easily learn it and it becomes easy to use this knowledge to attack the CAN. Another problem is the increase in traffic. For example, when messages are encrypted by AES, the date length must be 128 bits. AES is a block cipher and requires that data length is a minimum of 128bits. Because data length in frames is up to 64 bits in CAN, the length must be extended to use AES. Therefore, one additional message will be created. As a result, it has an effect in real time on the vehicle. If an important message cannot be transferred within a specific timeframe, this can affect the driver and cause an accident. Finally, another serious problem is that there is insufficient defense against a side-channel attack. Today, AES is considered a safe cryptography algorithm method. However, in a side-channel attack, AES becomes unsafe. In other words, even though AES is used for encrypting

messages in the CAN, confidentiality cannot be ensured.

2.3.5 Driver Behavior Based Methods

A method using driver behavior also suggested. The suggested method uses data based on driver behaviors such as angle of handle, speed of vehicle revolution, and position of pedal and so on. These data can give clue of driver behavior and it would be used for detecting attack. This method can be used with machine learning or data mining algorithm. [29] is one of the method using driver behavior and data mining. This method suggested framework that is tailored to vehicle. This method uses Decision Support Engine (DSE) that is for dealing with large scale traffic and detecting quickly attack. It consists of Device Status Analyzer, Device Status DB, Device Maintenance Manager, and Decision Engine Core. Also, There is Security Monitor that collects information of devices in vehicle through IDS. In this framework, Security Monitor send information of device state DSE, and then Device Status Analyzer in DSE analyzes information of device state using data mining algorithm. It classifies normal or abnormal based on analyzed information and these information are stored in Device Status DB. In this situation, detection rule is made or updated by data mining. The Security Monitor detects system event or abnormal traffic using updated detection rule. This idea can enhance detection performance in vehicle, but it can revise vehicle to adapt this framework and it is not be proved.

2.3.6 Security Gateway

Because there are an increased number of ECUs in the CAN bus nowadays, a separate domain is required to manage the corresponding increase in traffic. To do this, gateways were introduced in the CAN, and there are some gateways in the CAN bus. One gateway manages one domain, and it enables ECUs to exchange messages. Once ECUs send messages that have to head to another domain through the gateway, the gateway finds the appropriate domain and sends these messages to the gateway that manages the corresponding domain. In this manner, the gateways in the CAN bus are in charge of managing one domain and communicating with other domains. Figure 2.8 shows some gateways and its domain.

The security gateway adds security functions to the gateways. Normally, there are two things that the security gateway performs. First is authenticating each ECU. In this situation, security gateway authenticates whether ECU in CAN is valid or not. Second is detecting attacks in CAN. At this time,

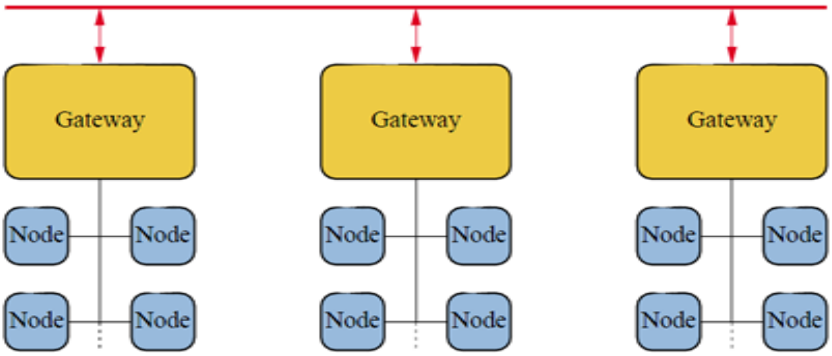

Figure 2.8: Gateway in CAN

security gateway detects attacks such as spoofing or DoS with specific algorithm. Figure 2.9 shows an example of security gateway.

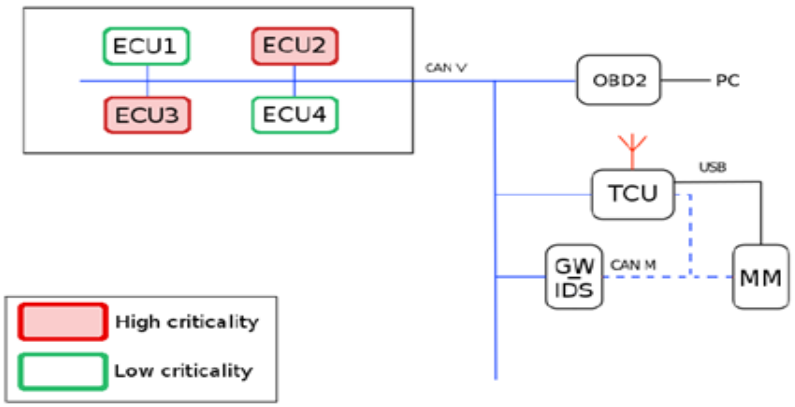

Figure 2.9: Security Gateway in CAN

Chapter 3. Approach

3.1 Assumption

Below are the required assumptions for the idea proposed in this paper.

1) The security gateway and the ECUs in the CAN bus already share a specific formula and key.

   - Basically, the manufacturer can insert anything in the ECU and gateway; therefore, they can insert a key into the ECU and gateway during the manufacturing process.

2) The IDs of the security gateway and the ECUs have already been determined.

   - It is also possible that the manufacturer can determine the ID of the ECU and gateway during the manufacturing process.

3) The security gateway stores the information about the ECU that exists in the domain.

   - During the manufacturing process, ECU information can be inserted into the security gateway by the manufacturer.

3.2 Attack Model

In this paper, attack model is defined as follow:

The attacker performs attacks using the On-Board Diagnostic-II (OBD-II) port. An OBD-II port is installed in vehicles for diagnostic purposes, and it is connected to the CAN bus. Thus, an attacker can attack the CAN bus by just connecting to the OBD-II port. Figure 3.1 and 3.2 show examples of vehicle attacks. Figure 3.1 shows an initial method and is somewhat complicated, but Figure 3.2 shows another, more convenient attack method. This is called CANtact and it makes attacks easy; because it is small, a driver cannot notice CANtact.

In this situation, the attacker can already know the ID of the ECUs in the CAN bus and the meaning of these IDs. As mentioned above, because CAN is a broadcasting environment, it is easy to

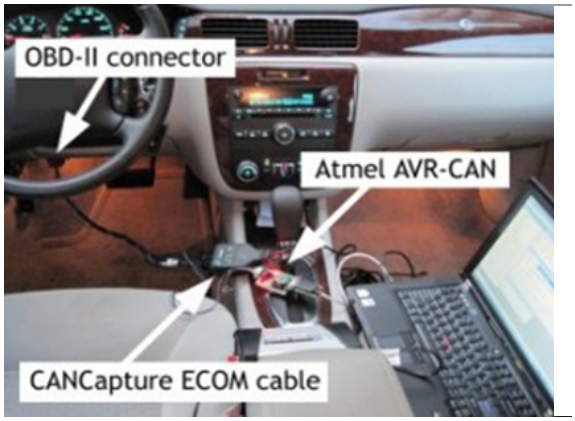

Figure 3.1: Example of attack method 1

Figure 3.2: Example of attack method 2

sniff messages. Therefore, the attacker can easily determine the IDs of ECUs. Furthermore, by analyzing the collected IDs, an attacker can determine the role of each ID in the vehicle. However, The attacker cannot know how the internal components of the vehicle are organized. As mentioned above, the attacker can know the ID of the ECUs in the CAN bus using the OBD-II port. However, by using this method, the attacker cannot know how the internal components of the vehicle are organized, because the attacker can only collect messages in the CAN.

There is another method attacking a vehicle. The attacker can insert malicious ECU into internal of vehicle. Although it is easy to install ECU in vehicle, the attacker can do it if he knows about the vehicle or through mechanic.Also, it is harder to detect attack ECU by driver, because it does not be shown to driver. In this situation, the attacker connects to malicious ECU using wireless device such as smart phone, tablet, and laptop, and so on and can easily know various information in CAN bus such as ID, messages, and so on. Based on this information, the attacker can do sniffing, spoofing, replay, DoS attack like attacking using OBD-II port. Therefore, the attacker can more easily cause hazardous situation than attacking using OBD-II. However, this attack method also cannot know all parts of vehicle and information of values such as key, formula that are used in defense, because the attacker cannot have information of structure of vehicle and ECU.

3.3 Defense

In this chapter, the security gateway that is the core of the idea will be described. Then, defending against a spoofing attack using the sequence of messages based on the driver's behavior will be explained.

Finally, a method of defending against DoS attacks using temporary IDs will be presented.

3.3.1 Security Gateway

In this paper, the security gateway basically monitors its domain and defends against spoofing and DoS attacks. Figure 3.3 shows the structure of the security gateway, which has four components. The role of each component is as follows:

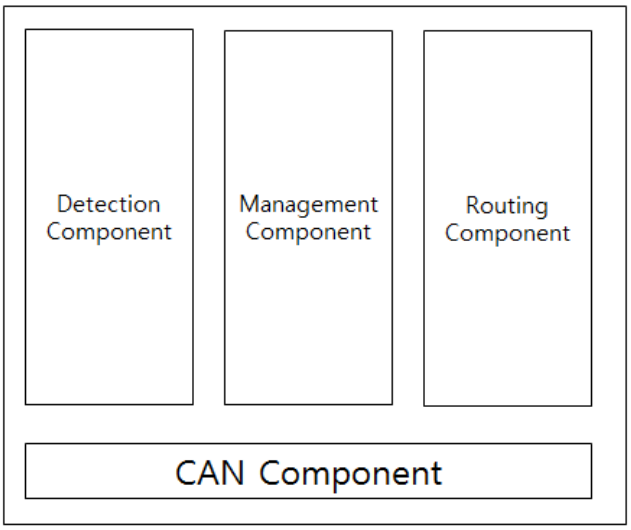

Figure 3.3: Structure of security gateway

1) Detection component

   - Component for detecting spoofing and DoS attacks.

2) Routing component

   - Component for arranging transfers to other domains.

3) Management component

   - Component for managing tables and creating messages and seeds

4) CAN component

   - Component for CAN communication

Figure 3.4 shows an example security gateway. As you can see, the security gateway manages its domain; it monitors the CAN bus in its domain and detects attacks.

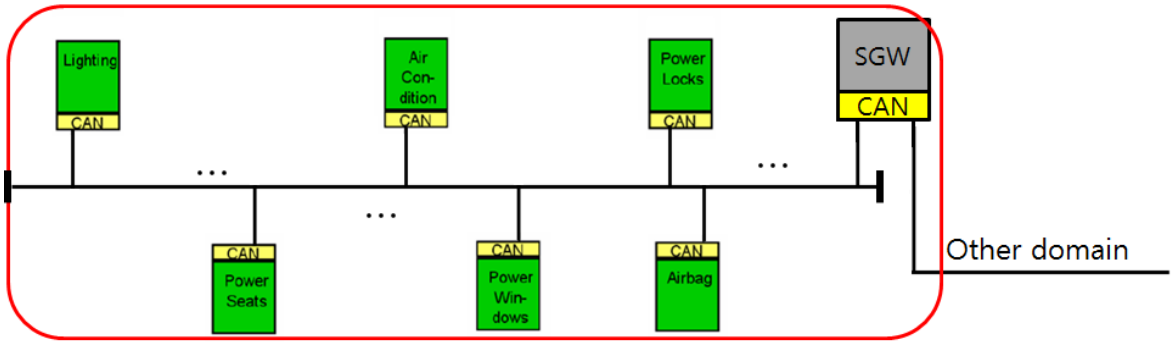

Figure 3.4: Example

3.3.2 Spoofing Defense

The sequence of messages based on the driver's behavior is used to defend against spoofing attacks. There are various ECUs in the CAN, and these are used for specific purposes. For example, different ECUs are in charge of the door, lights, brakes, engine, and so on.

This, like all ECUs is used for a specific purpose. We can think how each ECU is associated with the others. For example, if the driver turns left or right, the driver first uses the brakes, then the steering wheel. Finally, the accelerator will be operated. Thus, the brake ECU, steering wheel ECU, and acceleration ECU will be operated in a regular sequence, and messages related to these ECUs will flow in the CAN bus in this order. This means that the sequence of messages is determined by the driver's behavior. Thus, if sequence of messages is monitored, a spoofing attack can be detected. Because an attacker cannot know what ECUs exist in the CAN bus, the attacker cannot determine the flow of messages. Thus, when the attacker tries to use a spoofing attack, it violates the sequence of messages, which can be detected by monitoring the sequence of messages. Using this information, the table can be made according to what ECUs represent the basic ECUs used in the vehicle. The table 3.1 represents the basic ECUs that are used in vehicle. By using these ECUs, the driver can adopt various behaviors. These ECUs are related to the engine, brakes, light, and so on.

The table 3.2 shows driver behavior based on the basic ECU; there are 10 driver's behaviors. For example, if the driver stops his vehicle, the brakes, gears, and engine will operate in that order. Thus, the Electronic Brake Control Module ECU, Transmission ECU, and Engine Control Module ECU will operate in this order and the related messages will flow in the CAN bus. Using this characteristic, the detection component in the security gateway monitors the CAN bus. When the first message in the

Table 3.1: Basic ECUs

	ECU
①	Electronic Stability Control ECU
②	Transmission ECU
③	Engine Control Module ECU
④	Electronic Break Control Module ECU
⑤	Electric Power Steering ECU
⑥	Throttle ECU
⑦	Instrument Cluster ECU
⑧	Electric Parking Break ECU
⑨	Light Control Module ECU
⑩	Adaptive Front Lighting ECU

table is detected, the next message in table will be monitored. If the expected message is not witnessed and other message is discovered, it considers the possibility of a spoofing attack. However, because this message may be sent by a normal ECU, a verification process must be performed. At this time, the verification process will be performed through the management component and SipHash, which is one of the hash algorithms used in this process. The SipHash is a key-based hash algorithm that uses a simple operation such as Add-Rotate-Xor (ARX). It is also optimized about a short message, resulting in a 64-bit tag value. Therefore, it is suitable for constrained devices such as an ECU.

Table 3.2: Sequence of messages based on driver's behavior

Driver's behavior	**Sequence of messages**
1) Usual	① → ② → ③
2) Left/Right turn, U-tern	④ → ② → ⑤ → ⑥ → ③
3) Ignition	⑥ → ③ → ② → ⑥ → ⑦
4) Stop	④ → ⑥ → ③ → ②
5) Acceleration	② → ⑥ → ③ → ⑦
6) Deacceleration	④ → ⑥ → ③ → ⑦
7) Parking/stop	④ → ② → ⑥ → ③ → ⑧ → ⑦
8) Light	⑨ → ⑩ → ⑦
9) Backward movement	④ → ② → ⑤ → ⑦ → ②
10) Change of line	⑨ → ⑦ → ⑤ → ⑥ → ③

The security gateway requests the hashed value of the suspect message. Sending the ECU that sent the suspect message calculates the hashed value and sends it to the security gateway. Then, the management component in the security gateway calculates the hashed value of the suspect message and compares it to the received hashed value. If the two hashed values are the same, verification succeeds, and if the hashed values are different, verification fails. In other words, it can determine that there has been an attack. At this time, a specific key is used in the verification process. As mentioned in the assumption, this key has already been shared between the security gateway and the ECUs. Because a

valid ECU can calculate a hashed value using its key, it can be verified. The attacker does not know this key, and so cannot create a hashed value. Thus, the attacker cannot send a verification message and the attack can be determined.

3.3.3 DoS Defense

Finally, a temporary ID for defending against DoS attacks is proposed. First, the security gateway monitors frames in the CAN bus and detects a DoS attack. A DoS attack is detected using the existing method that analyzes the frequency of messages. At this time, messages that have high priority are monitored. If a monitored message is faster than its frequency, it may be a DoS attack. To defend against this, the management component in the security gateway creates a seed for a temporary ID. Then, the temporary ID used in each ECU is created using a seed in the management component before it is sent and examined to determine whether they have the same ID. If the ID is the same, the seed is created again and the process is repeated. In addition, once the temporary ID used in each ECU is created, the table that corresponds to the existing ID will be created for smooth communication with the other domains.

The created seed is transferred to each ECU. The ECUs that receive messages from the security gateway make a temporary ID using the transferred seed. At this time, the already shared formula and SipHash are used to create a temporary ID. First, the median value that is used in the SipHash input is made using the seed. Below is the formula for determining the median value.

$$\frac{ID + seed}{n} \tag{3.1}$$

N represents the total number of ECUs in a domain and seed is value from security gateway. ID is ECU's ID. Basically, because an attacker cannot know how many ECUs there are in a domain, the attacker does not know n. Thus, this formula is appropriate. Next, below is the final temporary ID.

$$TemporaryID : Lowest\ 11bit\ of\ SipHash(\frac{ID_1 + seed}{n})\ mod\ ID_2 \tag{3.2}$$

ID_1 is ECU's ID and N is the total number of ECUs in domain. Seed is value from security gateway and (ID_2) is ID that has highest priority in domain. The median value is inserted into SipHash as

input. Then, a 64-bit tag value is obtained and the 11 bit that is the lowest bit in the 64-bit tag value is extracted. A modular operation is performed with this value and the highest ID. Through modular operation, the value that is lower than highest ID will be obtained, and it means that this value has a higher priority than the highest ID in the CAN bus. Therefore, if ECUs use the replaced temporary ID instead of the original ID, there is no communication problem because the temporary ID has a higher priority than the attacker's ID. This enables ECUs to communicate with each other regardless of any DoS attack. At this time, the attacker stops their DoS attack because he knows the attack is having no effect on the CAN bus. If so, the security gateway detects this and sends messages that notify other components that the DoS attack has ended. Then, the table for communicating with other domains is abolished. At this time, an attacker can use the temporary ID when trying to perform a DoS attack. However, because the temporary ID is randomly created, the attacker cannot use the previous temporary ID. Therefore, there is no problem.

For example, if there is a DoS attack such as in figure 3.5, the security gateway detects the DoS attack and creates a seed, which is then transferred to each ECU.

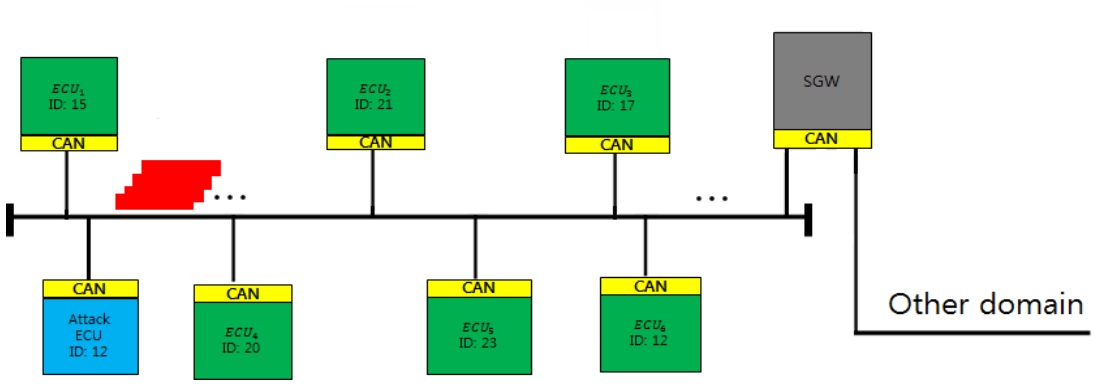

Figure 3.5: Example of DoS attack in CAN

The ECUs that receive messages from the security gateway make a temporary ID using the seed and exchange the original ID for the temporary ID. Figure 3.6 shows such a situation where because the attacker's ID has a lower priority than the temporary IDs, the DoS attack is ineffective. In this situation, attacker cannot know temporary ID and make it, because attacker does not have key that is used in SipHash. Thus, the attacker stops their DoS attack, which the security gateway detects. The security gateway then sends messages that notify other components that the DoS attack has finished, and the ECUs return to their original IDs. If DoS attack occurs again, defense against it will be performed. At

this time, temporary ID is new because seed is new. Therefore, DoS attack will be ineffective.

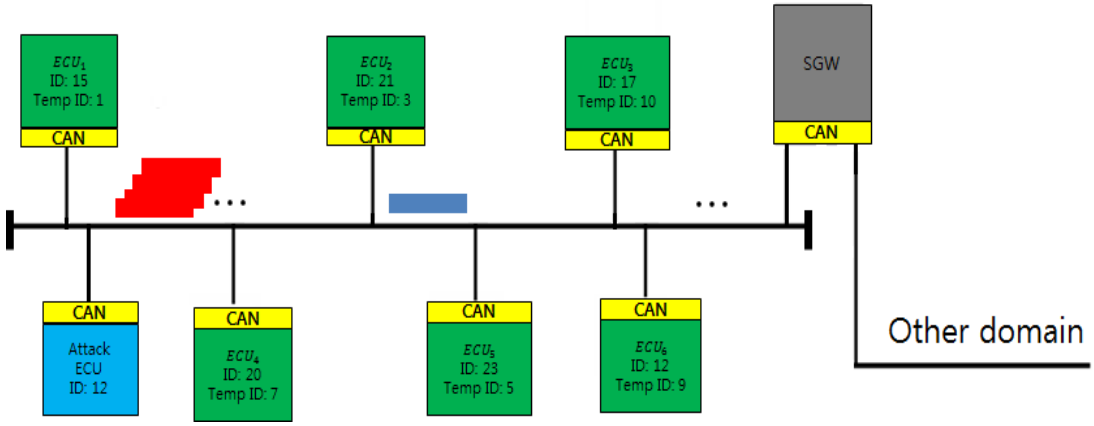

Figure 3.6: Defense using temporary ID

Chapter 4. Implemetation

4.1 OMNeT++

OMNeT++, which is a network simulator, was used to verify proposed idea. Briefly, we want to introduce OMNeT++. OMNeT++ is an open source program that can test networks and is used for testing various network environments such as Ethernet, wireless, mobile, and p2p. Because OMNeT++ is based on C++, anyone who uses C++ can easily organize an experimental environment by modifying and testing code in OMNeT++. In addition, OMNeT++ is a module-based program, and there are various modules in OMNeT++. Normally, one module is used by combining it with other modules and calling it a component. Such components can then be used in a simulation. For example, if someone wants to test a simulation of wireless devices, modules that can do wireless communication, application modules, and other ancillary modules are required to test the simulation. By combining these modules, wireless components can be made and used in a wireless simulation. Figure 4.1 shows the IPv6 model that OMNeT++ offers as a basic example.

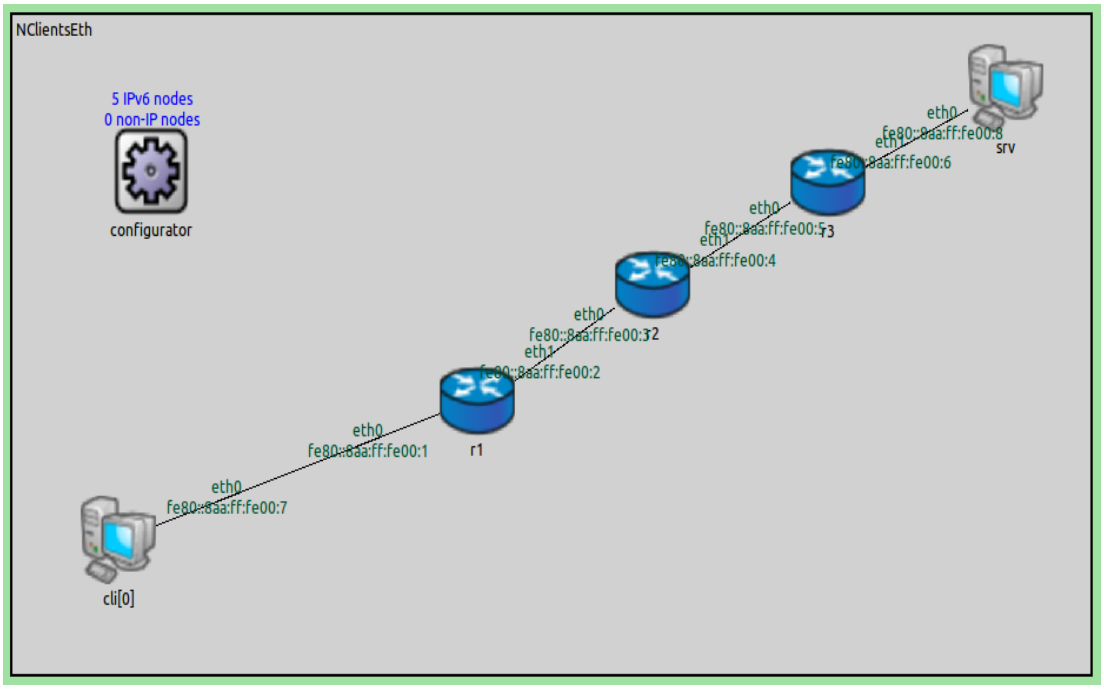

Figure 4.1: IPv6 model in OMNeT++

4.2 Experiment Environment

As mentioned above, OMNeT++ offers various basic models for network simulation. However, CAN models are not offered in OMNeT++. Therefore, Keigo Keigo *et al* [31] developed a CAN model based on the basic model in OMNeT++ and offered it to people. Figure 4.2 shows two simple examples implemented using the CAN model.

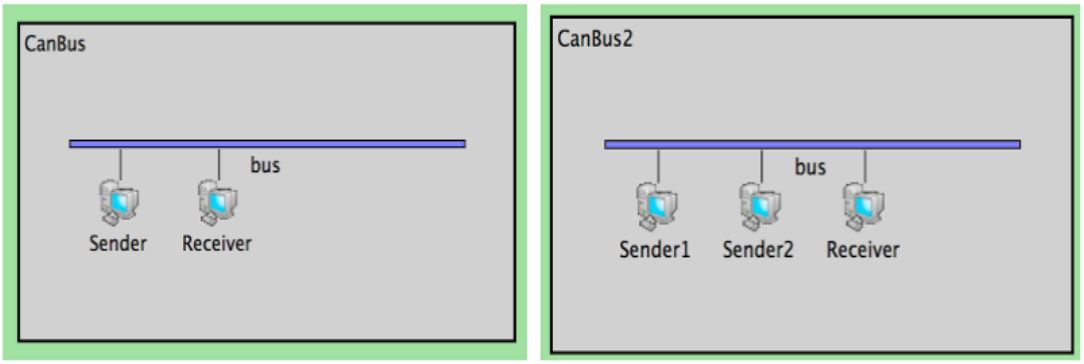

Figure 4.2: Examples of CAN model

However, the CAN basic model that is offered by [31] cannot simultaneously send and receive messages, because sender and receiver are separated. Thus, it is different to actual ECUs that can send and receive messages simultaneously. In addition, because there is only the data frame, additional implementation is required to test ideas fully. Accordingly, the CAN basic model was modified. Furthermore, the security gateway proposed in this paper was added to this CAN basic model along with some minor parts, which were added or modified.

Next, the basic topology was implemented based on the structure of the vehicle used in this experiment. Before explaining the basic topology, the structure of the vehicle requires explication. Figure 4.3 shows the structure of vehicle.

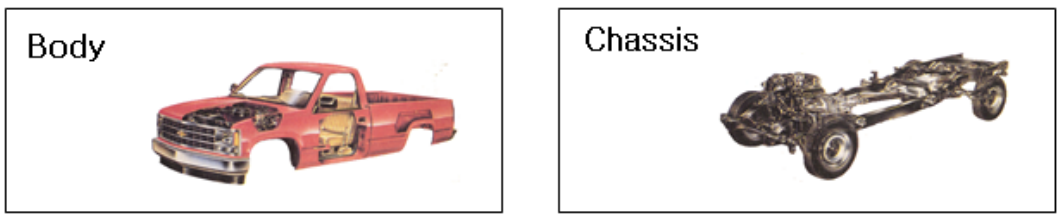

Figure 4.3: Structure of vehicle

The vehicle can be divided into the body and the chassis. The body is used when a driver and

passengers are onboard or cargo is carried and the chassis is needed when driving. Between the body and chassis, the chassis is an important part that makes up the interior of the vehicle. The chassis also consists of various parts. Figure 4.4 shows how chassis is composed.

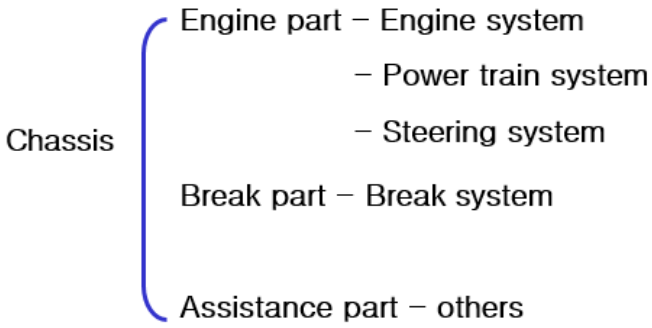

Figure 4.4: Configuration of chassis

As you can see, the chassis can be divided into the engine, brakes, and assistance parts, and there are various systems in each part.

A basic topology was created based on three parts. Each part was made up of ECUs in table, which is presented in chapter 3. The position of the ECUs could be determined based on figure 4.4. Figure 4.5 shows the basic topology, and as you can see, each of the three parts is made of the ECUs mentioned above. In addition, there are three security gateways and each security gateway manages its domain. For example, security gateway 1 monitors the engine part and detects spoofing and DoS attacks and both sends and receives messages from other domains.

Below is ID of basic ECUs that was used in experiment.

1) Engine part

- Throttle ECU: 0x61
- Engine Control Module ECU: 0x31
- Electric Power Steering ECU: 0x51
- Transmission ECU:0x21

2) Break part

- Electronic Break Control Module ECU:0x41
- Electric Parking Break ECU:0x81

3) Assistance part

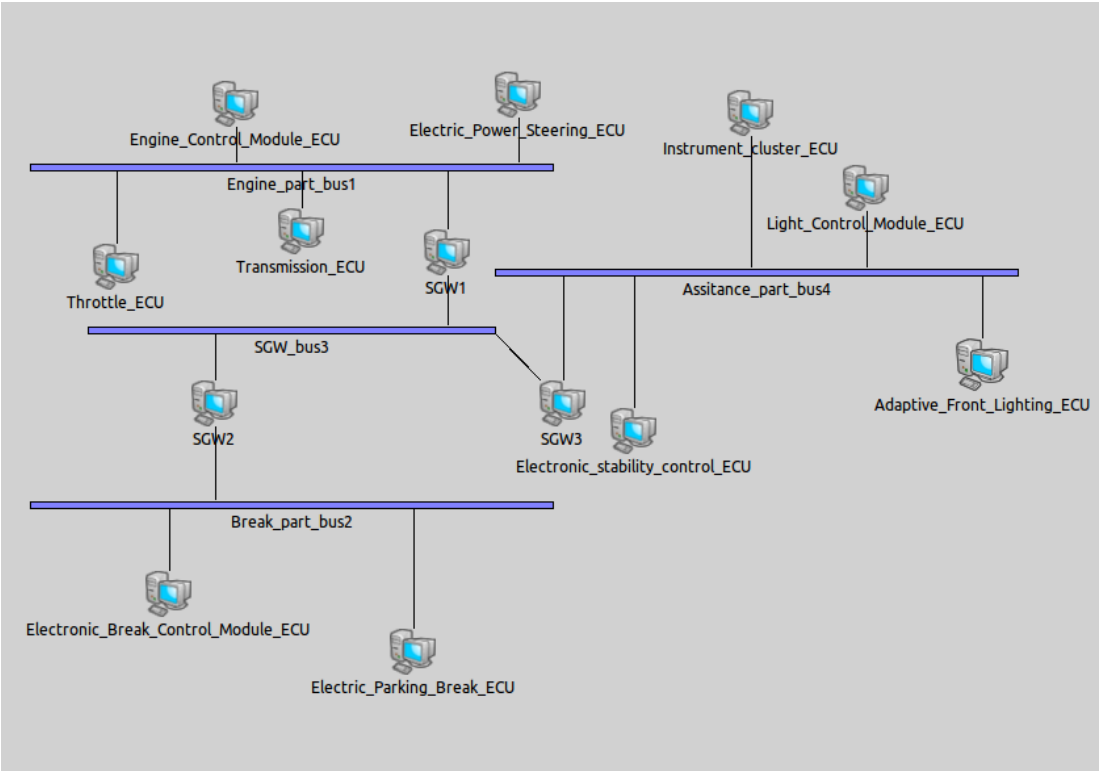

Figure 4.5: Basic topology

- Electronic Stability Control ECU:0x11
- Light Control Module ECU:0x91
- Adaptive Front Lighting ECU:0x101
- Instrument Cluster ECU:0x71

Two experiments were performed based on the basic topology. One simulated a defense against spoofing attacks, the other simulated a defense against a DoS attack. Additional ECUs were added to the basic topology in each experiment, 20 to the first experiment and 30 to the second. At this time, the IDs of the additional ECUs were randomly determined, and these IDs were larger than the basic ECUs so that their priorities were low.

Chapter 5. Result

5.1 Spoofing Defense

In this chapter, the result of defense against spoofing and DoS attacks is presented when the proposed idea was tested in an experimental environment in chapter 4. The attack ECU is in one of the basic topologies for the spoofing defense and the attack frame is transferred to the CAN bus at an arbitrary time. Two parts were measured in the spoofing attack. One is the detection rate, the other is the increase in traffic. The detection rate was measured by changing the number of attack frames, when there were 30 or 40 ECUs. The table 5.1 shows the detection rate for 30 and 40 ECUs.

Table 5.1: Detection rate

		Number of ECUs	
		30 ECUs	**40 ECUs**
Number of attack frames	**10**	99.3%	99.0%
	20	99.1%	98.6%
	30	98.7%	98.3%

Examining the table 5.1, the detection rate is approximately 99%, 98.8%, and 98.5% when there were 10, 20, and 30 attack frames; it generally shows a high detection rate. However, the detection rate becomes low as the number of ECUs and attack frames increases, because the lower detection rate is a false positive that detects normal frames as attack frames. In the proposed spoofing defense idea, there is a verification process and false positives occur in this process. If a frame that needs verification is sent to a security gateway, this frame may be considered a verified. Thus, it will fail the verification process and a false positive will occur despite it being a valid frame. The possibility of this phenomenon increases as the number of ECUs increases. Because the ECUs increase, the total number of frames in the CAN bus also increases and the possibility of a false positive will increase. Although false positives occur, the entire detection rate is about 98.8%, demonstrating a high detection rate.

Next, how much traffic increased was measured in the proposed idea. Table 5.2 and 5.3 show how much traffic increases when the total number of ECUs is 30 and 40.

The number of attack frames is 30 in this experiment. The rate represents the rate of valid frames in the total frames, and it means an increased number of frames. Examining each table, the increase was

Table 5.2: Increase of traffic (30 ECUs)

Driver's behavior	Number of total frames (A)	Number of valid frames (B)	Rate (B/A * 100 %)
1) Usual	4,060	14	**0.34%**
2) Left/Right turn, U-tern	13,522	20	**0.41%**
3) Ignition	20,143	19	**0.09%**
4) Stop	15,423	16	**0.10%**
5) Acceleration	13,696	19	**0.13%**
6) Deacceleration	11,776	13	**0.11%**
7) Parking/stop	27,748	24	**0.08%**
8) Light	5,611	17	**0.30%**
9) Backward movement	22,949	18	**0.10%**
10) Change of line	20,424	21	**0.10%**

at most 0.41% and at least 0.08%, when there were 30 ECUs. It shows an average increase of 0.11%. This is quite low considering the rate of total frames. When there were 40 ECUs, the increase was at most 0.37% and at least 0.11%. The average increase was 0.19%, thus it was somewhat increased. As the number of ECUs increased, the total frames similarly increased. According to this, the number of frames that required verification was relatively increased. Therefore, it is determined that the traffic increased slightly. However, the average increase was 0.11%, which accounted for a relatively small extra portion of total frames.

Table 5.3: Increase of traffic (40 ECUs)

Driver's behavior	Number of total frames (A)	Number of valid frames (B)	Rate (B/A * 100 %)
1) Usual	9,053	34	**0.37%**
2) Left/Right turn, U-tern	18,570	39	**0.21%**
3) Ignition	27,336	42	**0.17%**
4) Stop	22,554	38	**0.16%**
5) Acceleration	20,139	36	**0.17%**
6) Deacceleration	17,664	40	**0.22%**
7) Parking/stop	43,044	55	**0.12%**
8) Light	12,572	42	**0.33%**
9) Backward movement	44,457	50	**0.11%**
10) Change of line	39,028	47	**0.12%**

5.2 DoS Defense

Next is result of the defense against DoS attacks. The attack ECU tried to attack the CAN bus at an arbitrary time. The DoS attack was maintained during regular time and then stopped. This was one period of a DoS attack, and it occurred randomly. In addition, the attack times and last attack time were random.

Figure 5.1 and 5.2 show the frame drop rate at the ECU. Basically, because the priority of the

attack frames is higher than the other frames during a DoS attack, other frames cannot be transferred. Therefore, the ECU's frame drop rate is very large. Figure 5.1 shows this situation. Initially, we can see that the frame drop rate gradually becomes large in Figure 5.1, because the ECU cannot send frames to the CAN bus. After that, once a temporary ID was created through the proposed idea, the communication between ECUs became possible, because the temporary ID had a higher priority than the one used in the DoS attack. This meant that the frame drop rate became low, as shown in figure 5.1. After that, the DoS attack stops and the original ECU ID is recovered. Therefore, ECU communication is possible and there is only a small frame drop rate. Finally, once the DoS attack starts again, defense against DoS attacks was initiated. As a result, the frame drop rate increased and decreased in a similar pattern to before.

The increased frame drop rate due to two attempted DoS attacks is shown in figure 5.1 and the frame drop rate is decreased by the implementation of temporary IDs. In other words, the defense against DoS attacks is effective. However, there is a delay in the defense, when the DoS attack is detected and action is taken against it. This delay is because of time required to make a seed and temporary ID at each ECU. In addition, the security gateway must check whether it holds the same temporary ID. If there is the same temporary ID, a seed must be created again to avoid collisions between ECUs. Furthermore, because the ECU makes a temporary ID using the received seed, it takes some time. Due to these, there is a small delay. Thus, it is determined that the delay is not a problem.

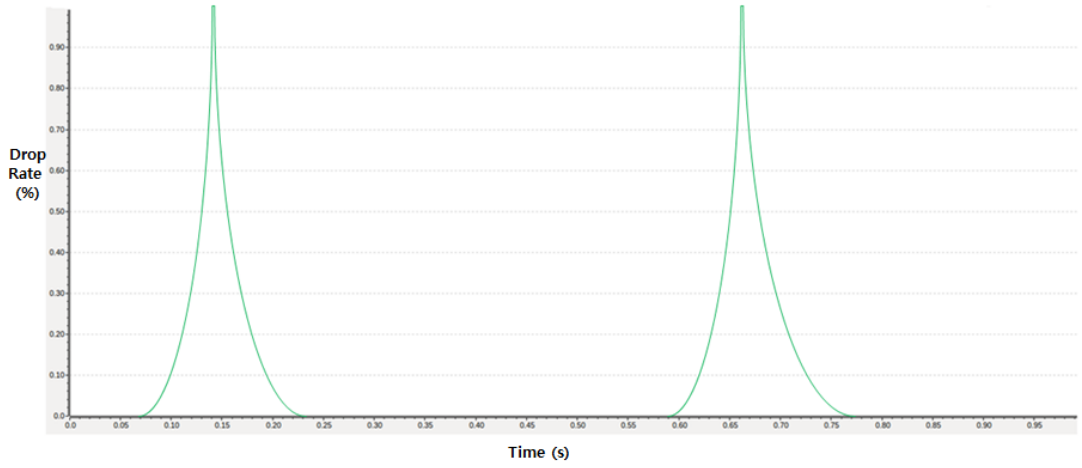

Figure 5.1: Frame drop rate 1

Figure 5.2 also shows that the defense against DoS attack was well performed. There are two DoS attacks and each trigger defenses. As in figure 5.1, the defense against DoS attack was effective. As

mentioned above, there is a difference in the attack time, because DoS attacks occur randomly. However, we can see a similar result to Figure 5.1.

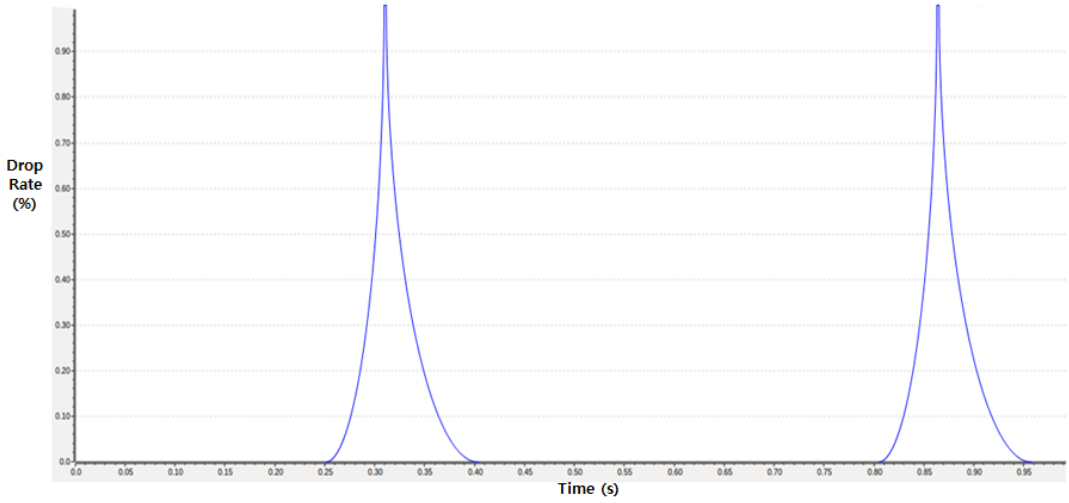

Figure 5.2: Frame drop rate 2

Chapter 6. Discussion

In this paper, the proposed defenses against spoofing and DoS attacks show some advantages. First, they present with a low traffic increase when defending against spoofing attack. The MAC- and cryptography-based methods had a problem of increased traffic. A small traffic increase will slightly affects the CAN, but a huge increase of traffic can be big problem for it. As mentioned above, the real time characteristic in vehicles is very important. If messages cannot be sent to a specific ECU at the proper time, it may effect the driver. Light and door messages have little effect on the driver; however, because engine and brake messages are vital to the driver's safety, these messages can have a huge effect on a driver. Considering that, any traffic increase is an important criterion when evaluating defense methods. In that part, the suggested idea shows a low increase of traffic and has little effect on the CAN bus.

Next, when defending against DoS attack, it shows that ECUs can communicate with each other during a DoS attack. Basically, ECUs have trouble communicating with each other during a DoS attack. Therefore, messages in the CAN bus cannot be handled at the proper time, which may harm the driver. As mentioned above, real time is very important when operating a vehicle. In a DoS attack, the CAN bus will be filled with the attacker's message, because the attacker's message has highest priority in the CAN bus. Due to this, other ECU messages cannot be sent to the CAN bus, causing a serious problem. By introducing a temporary ID, the effect of a DoS attack was mitigated. Through experimentation, the frame drop rate was decreased, meaning the ECUs could communicate with each other despite the DoS attack. However, there was a small delay time in the defense against DoS attack. The frame drop rate was considerably increased due to the delay time. Thus, reducing the delay time is required to decrease the effect of a DoS attack.

Finally, suggested idea in this paper has similarities in [29]. However, [29] is detection using anomaly behavior based on data that are collected in vehicle. In this situation, detection rule is always updated and it used for detect attack. It may be burden to vehicle. Although vehicle has been developed, it is not easy to dealing with large scale traffic in vehicle. To do this, vehicle must have more resource than previous one and it would cause increasing cost of vehicle. Also, it does not prove its efficiency and

effectiveness. On the other hand, suggested idea is detection based on signature using driver behavior. Because it stores table that is used for detecting attack in advance, it can not be burden of vehicle. Also, suggested idea proved its efficiency and effectiveness through experiment. Although experiment was performed using software, it is enough to show its efficiency and effectiveness.

Chapter 7. Conclusion

In this paper, defense against spoofing and DoS attack through a security gateway is proposed. In the case of defense against a spoofing attack, the sequence of messages is determined based on the driver's behavior, creating a table. Whether a spoofing attack occurs in the CAN bus is monitored using this table. In addition, via a verification process, valid frames are verified and malicious frames can be detected. In the case of a DoS attack, a temporary ID is created based on a seed. At this time, SipHash is used for calculating a temporary ID that has higher priority than an attack frame's priority. This enables ECUs to smoothly communicate with each other regardless of any DoS attack.

Through OMNeT++, experiments into defending against spoofing and DoS attacks were performed. Based on basic topology, the detection rate in defending against spoofing attack was approximately 98.8%, and the traffic increase was at most 0.19%. This demonstrates that the proposed idea is efficient. In defense against a DoS attack, the proposed idea demonstrated that it could effectively defend against DoS attack by analyzing the frame drop rate.

Table 7.1 is a comparison of existing ideas with the proposed idea. By examining this table, the proposed idea solves problems that the existing ideas could not solve, such as increases in traffic volume, the simultaneous detection of spoofing and DoS attacks, etc.

There are some future works, the first of which is the enhancement of the detection rate. It may be possible to extend the table that is used in defending against spoofing attacks. By adding new items, the table will be more detailed, which will improve the detection rate. In addition, there will be more ECUs than in the current model. Thus, it is necessary to extend items in the table to use this idea in an environment in which more ECUs exist. Second, the verification process must be improved, to reduce the number of false positives. This can be solved by adding another verification process. Finally, reducing the defense time during the initial period of a DoS attack is necessary. The suggested idea takes time to begin defending against a DoS attack, causing the frame drop rate to reach approximately 100%. This can be reduced by applying a new algorithm that can detect a DoS attack more rapidly or by improving the process of generating a temporary ID.

Table 7.1: Comparison existing ideas with proposed idea

	Increase of traffic volume	**Detection rate**	**Overhead**	**Modification of CAN protocol**	**Key management**	**Defending soopfing & DoS attacks**	**P.O.C (Proof of Concept)**
Proposed idea	**$\simeq$ 0.11%**	**$\simeq$ 98.8%**	**Relatevely low**	**X**	**O**	**O**	**O**
Ujiie *et al* [16]	$\simeq$ 0%	$\simeq$ 99%	Relatevely low	X	X	X	O
Matumoto *et al* [17]	None	None	None	X	X	X	X
Hartkopp *et al* [27]	None	None	None	O	O	X	X
Kubota *et al* [20]	$\simeq$ 50%	None	High	X	O	X	O

References

[1] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, and S. Savage, (2010, May). "Experimental security analysis of a modern automobile," In Security and Privacy (SP), 2010 IEEE Symposium on (pp. 447-462). IEEE.

[2] S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, and T. Kohno, (2011, August). "Comprehensive Experimental Analyses of Automotive Attack Surfaces," In USENIX Security Symposium.

[3] I. Roufa, R. Miller, H. Mustafaa, T. Taylora, O. Sangho, W. Xu, M. Gruteserb, W. Trappeb, , and I. Seskarb, (2010, February). "Security and privacy vulnerabilities of in-car wireless networks: A tire pressure monitoring system case study," In 19th USENIX Security Symposium, Washington DC (pp. 11-13).

[4] M. Nakano, T. Matsumoto, C. Vuillaume, and M. Kotani, "*Automotive information security: Threats and Countermeasures of ECU · automotive LAN · outside network*," Nikkei BP, Tokyo, 198 pages. *(in Japanese)*

[5] S. Jason. "How to Hack Your Mini Cooper: Reverse Engineering CAN Messages on Passenger Automobiles," DEF CON 21 (2013).

[6] Hyeryun Lee, Kyoungjin Kim, Gihyun Jung, Kyunghee Choi, "Research of generate a test case to verify the possibility of external threat of the automotive ECU," Journal of The Korea Society of Computer and Information, Vol. 18, No. 9, pp.21 – 31, 2013. 09

[7] Hyeryun Lee, Kyoungjin Kim, Gihyun Jung, Kyunghee, Seungkyu Park, Dokeun Kwon, "Studies of the possibility of external threats of the automotive ECU through simulation test environment," Journal of The Korea Society of Computer and Information, Vol. 18, No. 11, pp.39 – 29, 2013. 11

[8] K. Pierre, T. Olovsson, and E. Jonsson. "Security aspects of the in-vehicle network in the connected car," Intelligent Vehicles Symposium (IV), 2011 IEEE. IEEE, 2011

[9] I. Studnia, V. Nicomette, E. Alata, Y. Deswarte, M. Kaâniche, and Y. Laarouchi, (2013, September). "Security of embedded automotive networks: state of the art and a research proposal," In SAFECOMP 2013 Workshop CARS (2nd Workshop on Critical Automotive applications: Robustness & Safety) of the 32nd International Conference on Computer Safety, Reliability and Security (p. NA).

[10] Kwangjo Kim, Dongsu Lee, "A Trend Study on Vehicle Security in Conuntries using escar Conference," Journal of The Korea Institute of Information Security and Cryptology Vol. 24, No. 2, pp.7 – 20, 2014. 4

[11] D. K. Nilsson, U. E. Larson, and E. Jonsson. "Efficient in-vehicle delayed data authentication based on compound message authentication codes," Vehicular Technology Conference, 2008. VTC 2008-Fall. IEEE 68th. IEEE, 2008.

[12] C. W. Lin, and A. Sangiovanni-Vincentelli. "Cyber security for the Controller Area Network (CAN) communication protocol." Cyber Security (CyberSecurity), 2012 International Conference on. IEEE, 2012.

[13] B. Groza, S. Murvay, A. van Herrewege, and I. Verbauwhede, (2012). "Libra-can: a lightweight broadcast authentication protocol for controller area networks," In Cryptology and Network Security (pp. 185-200). Springer Berlin Heidelberg.

[14] C. Szilagyi, and P. Koopman, (2010, October). "Low cost multicast authentication via validity voting in time-triggered embedded control networks," In Proceedings of the 5th Workshop on Embedded Systems Security (p. 10). ACM.

[15] J. Yajima, M. Takenake, and T. Hasebe, "White-List CAN Hub with Disable Functionality against Attack Message," 2015 Symposium on Cryptography and Information Security (SCIS 2015), Jan. 20-23, 2015, Fukuoka, Japan. *(in Japanese)*

[16] Y. Ujiie, T. Kishikawa, T. Haga, and H. Matsushima, "Proposal of CAN Filtering Technology for In-Vehicle Network," Symposium on Cryptography and Information Security, 2015. *(in Japanese)*

[17] T. Matsumoto, M. Hata, M. Tanabe, K. Yoshioka, and K. Oishi "A method of preventing unauthorized data transmission in controller area network," Vehicular Technology Conference (VTC Spring), 2012 IEEE 75th. IEEE, 2012.

[18] M. Muter, A. Groll, and F. C. Freiling. “A structured approach to anomaly detection for in-vehicle networks,” Information Assurance and Security (IAS), 2010 Sixth International Conference on. IEEE, 2010.

[19] O. Satoshi, and I. Tasuku, “Intrusion Detection for In-vehicle Networks without Modifying Legacy ECUs,” IPSJ SIG Technical Report 2013. *(in Japanese)*

[20] T. Kubota, M. Nakano, R. Kurachi, S. Honda, M. Shiozaki, and T. Fujuno, “The demonstration system of encrypted CAN communication and the side-channel attack evaluation,” 2015 Symposium on Cryptography and Information Security (SCIS 2015), Jan. 20-23, 2015, Fukuoka, Japan. *(in Japanese)*

[21] T. Kishikawa, Y. Ujiie, T. Haga, H. Matsushima, M. Tanabe, Y. Kitamuram and J. Anzai, “Proposal of Security CAN to Protect In-Vehicle Network: Updatable CAN Protection Method using HW/SW Cooperation and Evaluation of the Method,” 2015 Symposium on Cryptography and Information Security (SCIS 2015), Jan. 20-23, 2015, Fukuoka, Japan. *(in Japanese)*

[22] T. Haga, Y. Ujiie, T. Kishikawa, H. Matsushima, M. Tanabe, Y. Kitamura, and J. Anzai, “Proposal of Security ECU to Protect In-Vehicle Network: Concept of CAN Protection Method that Suppresses the Introduction Impact,” 2015 Symposium on Cryptography and Information Security (SCIS 2015), Jan. 20-23, 2015, Fukuoka, Japan. *(in Japanese)*

[23] M. Tanabe, Y. Kitamura, J. Anzai, T. Kishikawa, Y. Ujiie, T. Haga, and H. Matsushima, “A Secure Switching Method between Monitoring Mode and Verifying Mode for In-Vehicle Network,” 2015 Symposium on Cryptography and Information Security (SCIS 2015), Jan. 20-23, 2015, Fukuoka, Japan. *(in Japanese)*

[24] N. Morita, K. Hakuta, T. Owada, and M. Kayashima, “A Proposal of a Message Authentication Method for Automotive Net-works,” 2015 Symposium on Cryptography and Information Security (SCIS 2015), Jan. 20-23, 2015, Fukuoka, Japan. *(in Japanese)*

[25] R. Kurachi, H. Takada, H. Ueda, and S. Horihata, “The proposal of the monitoring system for in-vehicle networks,” 2015 Symposium on Cryptography and Information Security (SCIS 2015), Jan. 20-23, 2015, Fukuoka, Japan. *(in Japanese)*

[26] M. Müter, and N. Asaj, (2011, June). “Entropy-based anomaly detection for in-vehicle networks,” In Intelligent Vehicles Symposium (IV), 2011 IEEE (pp. 1110-1115). IEEE.

[27] O. Hartkopp, C. Reuber, and R. Schilling, (2012, November). “MaCAN-message authenticated CAN,” In 10th Int. Conf. on Embedded Security in Cars (ESCAR 2012).

[28] A. Hazem, and H. A. H. Fahmy, (2012, November). “LCAP-A Lightweight CAN Authentication Pro-tocol for securing in-vehicle networks,” In 10th escar Embedded Security in Cars Conference, Berlin, Germany (Vol. 6).

[29] Byungil Kwak, Miran Han, Ahreum Kang, Huykang Kim, “A study on detection methodology of threat on cars from the viewpoint of IoT,” Journal of The Korea Institute of Information Security and Cryptology, Vol. 25, No. 2, pp.411 – 421, 2015. 4

[30] A. Jean-Philippe, and D. J. Bernstein. “SipHash: a fast short-input PRF,” Progress in Cryptology-INDOCRYPT 2012. Springer Berlin Heidelberg, 2012. 489-508

[31] K. Keigo, Y. Matsubara, and H. Takada. “A Simulation Environment and preliminary evaluation for Automotive CAN-Ethernet AVB Networks,” arXiv preprint arXiv:1409.0998 (2014)

Appendices

A Source Code

Security Gateway Code

```
#include <stdio.h>
#include "CanSecurityGateway.h"
#include "siphash.h"
#include <time.h>
#include <exception>

Define_Module(CanSecurityGateway);

simsignal_t CanSecurityGateway::DetectionSignal = SIMSIGNAL_NULL;
simsignal_t CanSecurityGateway::ValidSignal = SIMSIGNAL_NULL;

static cEnvir& operator<<(cEnvir& out, cMessage *msg)
{
    out.printf("(%s)%s", msg->getClassName(), msg->getFullName());
    return out;
}

CanSecurityGateway::CanSecurityGateway()
{

}

CanSecurityGateway::~CanSecurityGateway()
{
}

CanComponent::CanComponent()
{

}

CanComponent::~CanComponent()
{
}
```

```
CanRouting::CanRouting()
{

}

CanRouting::~CanRouting()
{
}

CanManagement::CanManagement()
{

}

CanManagement::~CanManagement()
{
}

CanDetection::CanDetection()
{

}

CanDetection::~CanDetection()
{
}

void CanSecurityGateway::initialize()
{

        numGates = gateSize("in");
        inputGateBaseId = gateBaseId("in");
        outputGateBaseId = gateBaseId("out");

        CanCominMes = new CanComponent();
        CanCominMes->initialize();

        CanDetinMes = new CanDetection();
        CanDetinMes->initialize(this);

        CanManinMes = new CanManagement();
        CanManinMes->initialize();

        CanRouinMes = new CanRouting();
```

```
        for(int i = 0; i < MAX_CAN_MESSAGE_ID; i++)
                isIn[i] = false;

        createRoutingTable(CanDetinMes);

        DetectionSignal = registerSignal("Detection");
    ValidSignal = registerSignal("Valid");
        packetsReceived = 0;

        WATCH(packetsReceived);
}

void CanDetection::initialize(CanSecurityGateway* hope)
{
        CanManinDet = new CanManagement();
        CanMesinDet = new CanSecurityGateway();
        CanRouinDet = new CanRouting();

        happy = true;
        isDetection = false;
        isDoS = false;
        DoSFirst = true;

        index = 0;



        CanManinDet->setMessageIDCollection();

    tagNode* temps = new tagNode;

    *root = new tagNode[5];

        root[0] = temps->CreateNode(0x41);
        root[1] = temps->CreateNode(0x11);
        root[2] = temps->CreateNode(0x61);
        root[3] = temps->CreateNode(0x21);
        root[4] = temps->CreateNode(0x91);

    tagNode* B = temps->CreateNode(0x21);
    tagNode* C = temps->CreateNode(0x61);
    tagNode* D = temps->CreateNode(0x31);
    tagNode* E = temps->CreateNode(0x51);
    tagNode* G = temps->CreateNode(0x61);
    tagNode* H = temps->CreateNode(0x61);
```

```
tagNode* I = temps->CreateNode(0x71);
tagNode* O = temps->CreateNode(0x71);
tagNode* J = temps->CreateNode(0x21);
tagNode* K = temps->CreateNode(0x31);
tagNode* P = temps->CreateNode(0x31);
tagNode* L = temps->CreateNode(0x81);
tagNode* M = temps->CreateNode(0x21);
tagNode* N = temps->CreateNode(0x71);

temps->AddChildNode(root[0], B);
temps->AddChildNode(root[0], C);
    temps->AddChildNode(B, E);
    temps->AddChildNode(B, G);
    temps->AddChildNode(C, D);
        temps->AddChildNode(E, H);
        temps->AddChildNode(E, I);
        temps->AddChildNode(D, J);
        temps->AddChildNode(D, O);
        temps->AddChildNode(G, K);
            temps->AddChildNode(H, P);
            temps->AddChildNode(K, L);
            temps->AddChildNode(I, M);
                    temps->AddChildNode(L, N);


tagNode* BB = temps->CreateNode(0x21);
tagNode* BC = temps->CreateNode(0x31);

temps->AddChildNode(root[1], BB);
    temps->AddChildNode(BB, BC);

tagNode* CB = temps->CreateNode(0x31);
tagNode* CC = temps->CreateNode(0x21);
tagNode* CD = temps->CreateNode(0x71);

temps->AddChildNode(root[2], CB);
    temps->AddChildNode(CB, CC);
            temps->AddChildNode(CC, CD);

tagNode* DB = temps->CreateNode(0x61);
tagNode* DC = temps->CreateNode(0x31);
tagNode* DD = temps->CreateNode(0x71);

temps->AddChildNode(root[3], DB);
    temps->AddChildNode(DB, DC);
```

```
                temps->AddChildNode(DC, DD);

    tagNode* EB = temps->CreateNode(0x101);
    tagNode* EC = temps->CreateNode(0x71);
    tagNode* ED = temps->CreateNode(0x71);
    tagNode* EF = temps->CreateNode(0x51);
    tagNode* EG = temps->CreateNode(0x61);
    tagNode* EH = temps->CreateNode(0x31);

    temps->AddChildNode(root[4], EB);
    temps->AddChildNode(root[4], EC);
        temps->AddChildNode(EB, ED);
        temps->AddChildNode(EC, EF);
                temps->AddChildNode(EF, EG);
                        temps->AddChildNode(EG, EH);



}

void CanComponent::initialize()
{
        CanDetinCom = new CanDetection();
        CanRouinCom = new CanRouting();
        CanManinCom = new CanManagement();

        CanDetinCom->setPeriodic(0);

}

void CanManagement::initialize()
{
        msbackup = NULL;

        CanCominMan = new CanComponent();

        already = false;
        AlreadyCheck = new CanFrame();
        AlreadyCheck->setMessageID(0);
        numDetected = 0;

}

void CanSecurityGateway::printId()
{
```

```
}

void CanSecurityGateway::handleMessage(cMessage *msg)
{
        CanCominMes->handleMessage(msg,this);
}

void CanSecurityGateway::record(bool happy,CanFrame* Check)
{
        if(!happy)
                emit(ValidSignal,Check);
        else
                emit(DetectionSignal,Check);
}

void CanComponent::handleMessage(cMessage *msg,CanSecurityGateway* hope)
{
        EV << "CanComponent: message " << msg << " received\n";

        if (!msg->isSelfMessage()) {
                CanTraffic *can_msg = check_and_cast<CanTraffic *>(msg);
                switch (can_msg->getType()) {
                case CAN_MESSAGE:

        if((hope->CanManinMes->AlreadyCheck->getMessageID() != 0) &&
        check_and_cast<CanFrame*>(msg)->getMessageID() == hope->CanManinMes->msbackup->getMessageID()){
                CanManinCom->VerifyMessage(check_and_cast<CanFrame*>(msg),hope);
        } else if((hope->CanManinMes->AlreadyCheck->getMessageID() != 0) &&
        check_and_cast<CanFrame*>(msg)->getMessageID() != hope->CanManinMes->msbackup->getMessageID()){

        }
                else{

                 if(!hope->CanDetinMes->isDoS && !hope->CanDetinMes->isDetection){
                        CanDetinCom->DoSDetection(check_and_cast<CanFrame*>(msg),hope);
                        CanDetinCom->SpoofingDetection(check_and_cast<CanFrame*>(msg),hope);
                }else if(hope->CanDetinMes->isDoS && !hope->CanDetinMes->isDetection){
                        CanDetinCom->DoSDetection(check_and_cast<CanFrame*>(msg),hope);
                }else if(!hope->CanDetinMes->isDoS && hope->CanDetinMes->isDetection){
                        CanDetinCom->SpoofingDetection(check_and_cast<CanFrame*>(msg),hope);
                }

                        }
                        break;
```

```
                case CAN_REMOTE:
                        handleRometeMessage(check_and_cast<CanRemoteFrame*>(msg), hope);
                        break;
                default:
                        error("Message with unexpected message type %d", can_msg->getType());
                }
        } else {
                EV << "CanSecurityGateway: Self-message " << msg << " received\n";
        }
}

void CanComponent::handleRometeMessage(CanRemoteFrame *msg,CanSecurityGateway* hope)
{
        uint messageid;
        messageid = check_and_cast<CanRemoteFrame*> (msg)->getMessageID();

        EV<<"Remote ID: "<<messageid<<endl;

        if(hope->isIn[messageid]){

                        cGate* ogate;

                        if(hope->messageIDs == 1){
                         ogate = hope->gate(hope->outputGateBaseId + 1);
                         send_message(hope, ogate, msg);
                        }else if(hope->messageIDs == 2){
                         ogate = hope->gate(hope->outputGateBaseId + 0);
                         send_message(hope, ogate, msg);
                        }else if(hope->messageIDs == 3){
                         ogate = hope->gate(hope->outputGateBaseId + 0);
                         send_message(hope, ogate, msg);
                        }

         }

        delete msg;

}

void CanComponent::send_message(CanSecurityGateway* hope, cGate* ogate, CanRemoteFrame *msg)
{
         if (ogate->isConnected() && !hope->CanManinMes->already) {
                cMessage *msg2 = msg->dup();
                hope->send(msg2, ogate);
        }
```

```
}

void CanRouting::handleCanMessageFromBus(CanFrame *msg,CanSecurityGateway* hope)
{
        int i;
        EV << "CanRouting: Received CAN Frame " << msg->getName() << "\n";
        uint messageid;
        CanFrame* Newframe = new CanFrame();
        bool find = false;

        if(!hope->CanDetinMes->isDoS){
          messageid = msg->getMessageID();
          for (i = 0; i < hope->numGates; i++) {
                  if (hope->routingTable[messageid][i]) {
                          EV<<"RoutingTable: "<<hope->routingTable[messageid][i]<<endl;
                          cGate* ogate = hope->gate(hope->outputGateBaseId + i);

                          if (ogate ->isConnected()) {
                                  cMessage *msg2 = msg->dup();
                                  hope->send(msg2, ogate);

                          }
                  }
           }
        EV << "CanRouting: Sent CAN Message '" << msg->getName() << "'\n";
        delete msg;
        } else if(hope->CanDetinMes->isDoS){
                  uint tempmessageid = msg->getMessageID();

                  std::map<int,int>::iterator itr;
                  itr = hope->CanManinMes->IDtranslation.find(tempmessageid);
                  if(itr != hope->CanManinMes->IDtranslation.end()){
                          messageid = itr->second;
                          find = true;
                  }

                  for (i = 0; i < hope->numGates; i++) {
                  if(msg->getArrivalGateId() != (hope->inputGateBaseId + i) && find){
                   EV<<"I am Here!!!"<<endl;
                   if (hope->routingTable[messageid][i]) {
                           cGate* ogate = hope->gate(hope->outputGateBaseId + i);
                          Newframe = msg->dup();
                          Newframe->setMessageID(messageid);

                           if (ogate ->isConnected()) {
```

```
                                cMessage *msg2 = Newframe->dup();
                                hope->send(msg2, ogate);

                        }
                 }
                } else {
                        cGate* ogate = hope->gate(hope->outputGateBaseId + i);

                        if (ogate->isConnected()) {
                                cMessage *msg2 = msg->dup();
                                hope->send(msg2, ogate);

                        }
                }
         }
        EV << "CanRouting: Sent CAN Message '" << msg->getName() << "'\n";
        delete msg;

        }
}

void CanSecurityGateway::createRoutingTable(CanDetection* CaninDet)
{
        int i, j;

        routingTable = new bool*[MAX_CAN_MESSAGE_ID];
        for (i = 0; i < MAX_CAN_MESSAGE_ID; i++) {
                routingTable[i] = new bool[numGates];
        }

        root = (par("routingmap").xmlValue());
        const char *ids = root->getAttribute("ID");
        messageIDs = ToDec(ids);

        EV<<"ID: "<<messageIDs<<endl;

        cXMLElementList MessageInfoElements = root->getChildrenByTagName("MessageInfo");
        EV << "Elements : " << (int)MessageInfoElements.size() << endl;

        cXMLElementList DetectionID = root->getChildrenByTagName("DetectionInfo");
        cXMLElement *message = DetectionID[0];
        CaninDet->DetectionID = ToDec(message->getAttribute("ID"));

        int CGWApps = par("numCGWApps");
        int CanBuses = par("numCanBuses");
```

```
        int EthernetBuses = par("numEthernetBuses");
        for (i = 0; i < CGWApps; i++) {
                NGTable.insert( std::map<std::string, int>::value_type( "CGW", i ) ); /
        }

        for (i = 0; i < CanBuses ; i++) {
                char CanBusName[32];
                sprintf(CanBusName, "CAN%d", i+1);
                NGTable.insert(std::map<std::string, int>::value_type(CanBusName, CGWApps + i));
        }

        for (i = 0; i < EthernetBuses ; i++) {
                char EtherBusName[32];
                sprintf(EtherBusName, "Ethernet");
                NGTable.insert( std::map<std::string, int>::value_type( EtherBusName, CGWApps + CanBuses + i
) );
        }

        std::map<std::string,int>::iterator it = NGTable.begin();
        while(it != NGTable.end()) {
                EV << (*it).first << ":" << (*it).second << endl;
                ++it;
        }

        for (i = 0; i < MAX_CAN_MESSAGE_ID; i++) {
                for (j = 0; j < numGates; j++) {
                        routingTable[i][j] = false;
                }
        }

        for (i = 0; i < (int)MessageInfoElements.size(); i++) {
                const char *id = MessageInfoElements[i]->getAttribute("ID");
                unsigned long messageID = ToDec(id);

                cXMLElementList MessageInfo = MessageInfoElements[i]->getChildren();
                for (j = 0; j < numGates; j++) {
                        cXMLElement *MessageInfoChild = MessageInfo[j];
                        if (atoi(MessageInfoChild->getAttribute("Send")) == 1) {
                                int value;
                                const char *name = MessageInfoChild->getAttribute("Name");

                                std::map<std::string, int>::iterator itr;
                                itr = NGTable.find(name);

                                if (itr != NGTable.end()) {
```

```
                                        value = itr->second;
                                } else {
                                        EV << "cannot find value corresponding to " << name << endl;
                                }
                                routingTable[messageID][value] = true;
                        }
                }
        }

        char id[32];
        EV << "*****************************************\n";
        for (i = 0; i < MAX_CAN_MESSAGE_ID; i++) {
                sprintf(id," 0x%04x ",i);
                EV << "MessageID: " << i << "(" << id << ") | " ;
                for (j = 0; j < numGates; j++) {
                        EV << routingTable[i][j] << " ";
                }
                EV << endl;
        }
        EV << "*****************************************\n";
}

void CanDetection::SpoofingDetection(CanFrame *msg,CanSecurityGateway* hope)
{
     tagNode* test = new tagNode;
     bool tests, isEnd = false;

      if(!hope->CanDetinMes->isDetection){
        for(; hope->CanDetinMes->index < 5; hope->CanDetinMes->index++){
                if(test->Search(hope->CanDetinMes->root[index],
                &hope->CanDetinMes->temp,msg->getMessageID(),&isEnd))
                {
                        hope->CanDetinMes->isDetection = true;
                        CanRouinDet->handleCanMessageFromBus(msg,hope);

                        break;
                }
        }
                if(hope->CanDetinMes->index == 5)
                        hope->CanDetinMes->index = 0;

        }  else if(hope->CanDetinMes->isDetection && !hope->CanManinMes->already){

                tagNode* temps = hope->CanDetinMes->temp;
                tests = test->Search(temps,&hope->CanDetinMes->temp,msg->getMessageID(),&isEnd);
```

```
                if(tests && hope->CanDetinMes->happy)
                {
                        CanRouinDet->handleCanMessageFromBus(msg,hope);
                }
                else if(!tests && hope->CanDetinMes->happy)
                {

                                if(!hope->CanManinMes->already)
                                {
                                        hope->CanManinMes->CheckMessage(hope,msg);

                                        hope->CanDetinMes->happy = false;
                                }

                 }


                if(isEnd)
                {
                        hope->CanDetinMes->index = 0;
                }

        }

        if(!hope->CanDetinMes->isDetection && !hope->CanDetinMes->isDoS) {
                CanRouinDet->handleCanMessageFromBus(msg,hope);
        }

        delete test;
}

void CanDetection::DoSDetection(CanFrame *msg,CanSecurityGateway* hope)
{
        SimTime Diff = 0;

        if(hope->CanDetinMes->DoSFirst && msg->getMessageID() == hope->CanDetinMes->DetectionID){
                Highmsbackup = msg->dup();
                hope->CanDetinMes->DoSFirst = false;
                start = 0;
        }else if(!hope->CanDetinMes->DoSFirst && msg->getMessageID() == hope->CanDetinMes->DetectionID){
                Diff = msg->getCreationTime() - Highmsbackup->getCreationTime();
                Highmsbackup = msg->dup();
                if(Periodic > Diff && !hope->CanDetinMes->isDoS){
                        hope->CanDetinMes->isDoS = true;
```

```
                check = 0;
                hope->CanDetinMes->CanManinDet->CreateTemperalID(msg,hope);
            }else if(Periodic < Diff && !hope->CanDetinMes->isDoS){
                EV<<"Normal"<<endl;
            }
        }

}

void CanManagement::CheckMessage(CanSecurityGateway* hope, CanFrame* backup)
{
        int temp;
        CanRemoteFrame* CheckMessage = new CanRemoteFrame();

        CheckMessage->setType(0x05);
        CheckMessage->setIsSent(false);
        CheckMessage->setMessageID(backup->getMessageID());
        CheckMessage->setControl_field((1|(0x1<<6)));
        CheckMessage->setFrameByteLength(0);

        hope->CanManinMes->msbackup = backup->dup();

        hope->CanManinMes->AlreadyCheck->setMessageID(backup->getMessageID());

        EV<<"Check ID: "<<backup->getMessageID()<<endl;

        for (int i = 0; i < hope->numGates; i++) {
            if(backup->getArrivalGateId() != (hope->inputGateBaseId + i))
                continue;
            temp = i;

        }

            cGate* ogate = hope->gate(hope->outputGateBaseId + temp);

            if (ogate->isConnected() && !hope->CanManinMes->already) {
                EV<<"Remote Connected";
                cMessage *msg2 = CheckMessage->dup();
                hope->send(msg2, ogate);
                hope->CanManinMes->already = true;
            }

        delete CheckMessage;
}
```

```
void CanManagement::VerifyMessage(CanFrame* Check,CanSecurityGateway* hope)
{
        uint8_t com1[8], key[16], i[1] ={0,};

        for(int i = 0; i < 16; i++)
                key[i] = i;

        siphash(com1, i, 1, key);
        bool test = false;

        if(!memcmp(com1,Check->getData(),8)){

                hope->record(test,Check);
                packetsReceived++;
        }
        else
        {
                test = true;
                hope->record(test,Check);
                packetsReceived++;
                hope->CanDetinMes->happy = true;
                hope->CanDetinMes->isDetection = false;
                hope->CanDetinMes->index = 0;
                hope->CanManinMes->already = false;
                hope->CanManinMes->AlreadyCheck->setMessageID(0);
                hope->CanDetinMes->temp = NULL;
        }

}

void CanManagement::CreateTemperalID(CanFrame* msg, CanSecurityGateway* hope)
{
        uint8_t com1[8], id[8], key[16], midres[2];
        int temp = 0, temp2 = 0, Mid[10], index, seed;
        CanFrame* CheckMessage = new CanFrame();

        srand((unsigned int)time(NULL));

        if(hope->CanDetinMes->isDoS){
        bool isSame;
        int messageID[10];
        getMessageIDCollection(messageID);

                for(index = 0; index < 16; index++)
                        key[index] = index;
```

```
isSame = false;

do{
seed = (rand()%65534)+1;

for(int i = 0; i < 8; i++)
        com1[i] = 0;
        isSame = false;

        for(index = 0; index < 10; index++){

                Mid[index] = (int)(seed+messageID[index])/10;
        }

        for(index = 0; index < 2; index++)
                midres[index] = 0;

        for(index = 0; index < 10; index++){
                midres[1] |= (Mid[index]&255);
                midres[0] |= ((Mid[index]>>8)&255);
                siphash(com1, midres, 2, key);
                temp |=com1[7];
                temp2 |=(com1[6]&7);
                temp |=(temp2<<8);
                TemperalMessageIDCollection[index] = temp%33;
                temp = 0;
                temp2 = 0;
                midres[1] = 0;
                midres[0] = 0;
        }

        for(index = 0; index < 10; index++){
                        if(TemperalMessageIDCollection[index] == 0){
                                isSame = true;


                        }
        }


}while(isSame);

        do{
                isSame = false;
                for(index = 0; index < 9; index++){
                        for(int j = index+1; j < 10; j++){
```

```
                        if(TemperalMessageIDCollection[index] ==
                        TemperalMessageIDCollection[j]){
                                isSame = true;
                                TemperalMessageIDCollection[j] += 1;

                        }
                }
        }

}while(isSame);

for(index = 0; index < 10; index++){

hope->CanManinMes->IDtranslation.insert(std::map<int, int>::
value_type(TemperalMessageIDCollection[index],messageID[index]));

}

do{
        isSame = false;
        tempSGWID = 0;
        for(int i = 0; i < 8; i++)
                com1[i] = 0;

        com1[7] |= (seed&255);
        com1[6] |= ((seed>>8)&255);
        com1[5] = 1;

        tempSGWID = seed + 0x01;

        midres[1] |= (tempSGWID&255);
        midres[0] |= ((tempSGWID>>8)&255);
        siphash(id, midres, 2, key);

        temp |=id[7];
        temp2 |=(id[6]&7);
        temp |=(temp2<<8);
        tempSGWID = temp%33;

        if(tempSGWID == 0)
                isSame = true;

}while(isSame);

CheckMessage->setType(0x00);
```

```
        CheckMessage->setIsSent(false);
        CheckMessage->setMessageID(tempSGWID);
        CheckMessage->setData(com1);
        CheckMessage->setFrameByteLength(8);

         for (int i = 0; i < hope->numGates; i++) {
                if(msg->getArrivalGateId() != (hope->inputGateBaseId + i))
                        continue;

                cGate* ogate = hope->gate(hope->outputGateBaseId + i);
                setOutputGateID(hope->outputGateBaseId + i);
                if (ogate->isConnected()) {
                        cMessage *msg2 = CheckMessage->dup();
                        hope->send(msg2, ogate);
                }
        }


        } else if(!hope->CanDetinMes->isDoS) {

                com1[5] = 0;

                CheckMessage->setType(0x00);
                CheckMessage->setIsSent(false);
                CheckMessage->setMessageID(tempSGWID);
                CheckMessage->setData(com1);
                CheckMessage->setFrameByteLength(8);

                        cGate* ogate = hope->gate(getOutputGateID());
                                if (ogate->isConnected()) {
                                cMessage *msg2 = CheckMessage->dup();
                                hope->send(msg2, ogate);
                        }


        }

        delete CheckMessage;
}

void CanManagement::setMessageIDCollection()
{

        MessageIDCollection[0] = 0x21;
        MessageIDCollection[1] = 0x71;
```

```
        MessageIDCollection[2] = 0x41;
        MessageIDCollection[3] = 0x171;
        MessageIDCollection[4] = 0x181;
        MessageIDCollection[5] = 0x191;
        MessageIDCollection[6] = 0x44;
        MessageIDCollection[7] = 0x55;
        MessageIDCollection[8] = 0x66;
        MessageIDCollection[9] = 0x77;
}

void CanManagement::getMessageIDCollection(int* message)
{
        for(int i = 0; i < 10; i++)
                message[i] = MessageIDCollection[i];
}

void CanManagement::setOutputGateID(int Out)
{
        outputGateID = Out;
}
int CanManagement::getOutputGateID()
{
        return outputGateID;
}

void CanSecurityGateway::finish()
{
    simtime_t t = simTime();
    recordScalar("simulated_time", t);
}

unsigned long CanSecurityGateway::ToDec(const char str[ ])
{
   short i = 0;
   short n;
   unsigned long x = 0;
   char c;

   while (str[i] != '\0') {

       if ('0' <= str[i] && str[i] <= '9') {

           n = str[i] - '0';
           } else if ('a' <= (c = tolower(str[i])) && c <= 'f') {
```

```
            n = c - 'a' + 10;
            } else {

            printf("%s",str);
            exit(0);
        }
        i++;
        x = x * 16 + n;
    }
    return (x);
}

int CanSecurityGateway::getnumGates() const
{
        return numGates;
}

void CanSecurityGateway::setnumGates(int Gate)
{
        numGates = Gate;
}

int CanSecurityGateway::getinputGateBaseId() const
{
        return inputGateBaseId;
}

void CanSecurityGateway::setinputGateBaseId(int Input)
{
        inputGateBaseId = Input;
}

int CanSecurityGateway::getoutputGateBaseId() const
{
        return outputGateBaseId;
}

void CanSecurityGateway::setoutputGateBaseId(int Output)
{
        outputGateBaseId = Output;
}

void CanDetection::setPeriodic(SimTime Pe)
{
        Periodic = Pe;
```

```
}

SimTime CanDetection::getPeriodic()
{
        return Periodic;
}

tagNode* tagNode::CreateNode(int newData)
{
    tagNode* newNode = new tagNode();

    newNode->index = newData;
    newNode->leftChild = NULL;
    newNode->rightSibling = NULL;

    return newNode;

}

void tagNode::DestroyNode(tagNode* node)
{
    delete node;
}

void tagNode::DestroyTree(tagNode* root)
{
    if(root->rightSibling != NULL)
        DestroyTree(root->rightSibling);

    if(root->leftChild != NULL)
        DestroyTree(root->leftChild);

    root->rightSibling = NULL;
    root->leftChild = NULL;

    DestroyNode(root);

}

void tagNode::AddChildNode(tagNode* parent, tagNode* child)
{
    if(parent->leftChild == NULL)
        parent->leftChild = child;
    else
    {
```

```
        tagNode* tempNode = parent->leftChild;
        while(tempNode->rightSibling != NULL)
            tempNode = tempNode->rightSibling;

        tempNode->rightSibling = child;
    }

}

void tagNode::PrintLevel(tagNode* node, int level)
{
    int depth = 0;
    tagNode* tempChild = node;
    tagNode* tempParent = node;

    while(depth <= level)
    {
        if(depth == level)
        {
            while(tempChild != NULL)
            {
                EV<<"ID: "<<tempChild->index;
                tempChild = tempChild->rightSibling;
            }

            if(tempParent->rightSibling != NULL)
            {
                tempParent = tempParent->rightSibling;
                tempChild = tempParent->leftChild;
            }

            else
                break;
        }

        else
        {

            tempParent = tempChild;
            tempChild = tempChild->leftChild;
            depth++;
        }
    }
```

```
    EV<<endl;

}

void tagNode::PrintTree(tagNode* node, int depth)
{
    int i = 0;
    for(i = 0; i < depth; i++)
        EV<<" ";


    if(node->leftChild != NULL)
        PrintTree(node->leftChild, depth + 1);

    if(node->rightSibling != NULL)
        PrintTree(node->rightSibling, depth);

}

bool tagNode::Search(tagNode* Parentnode, tagNode** temp, int ID, bool* isEnd)
{
        tagNode* tempParent = Parentnode;
        bool find = false;

        while(tempParent != NULL)
        {
                if(tempParent->index == ID){

                        if(tempParent->leftChild!=NULL)
                        {
                                *temp = tempParent->leftChild;
                                find = true;
                        }else
                        {
                                *isEnd = true;
                        }
                        break;
                }
                tempParent = tempParent->rightSibling;
        }


        return find;

}
```

ECU Code

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>

#include "CanECU_Base.h"
#include "CanApp_m.h"
#include "CanFrame_m.h"
#include "siphash.h"
#include <time.h>

Define_Module(CanECU_Base);

simsignal_t CanECU_Base::sentPkSignal = SIMSIGNAL_NULL;
simsignal_t CanECU_Base::rcvdPkSignal = SIMSIGNAL_NULL;
simsignal_t CanECU_Base::messageIDSignal = SIMSIGNAL_NULL;
simsignal_t CanECU_Base::DataSignal = SIMSIGNAL_NULL;
simsignal_t CanECU_Base::sendermessageIDSignal = SIMSIGNAL_NULL;
simsignal_t CanECU_Base::sendertempmessageIDSignal1 = SIMSIGNAL_NULL;
simsignal_t CanECU_Base::sendertempmessageIDSignal2 = SIMSIGNAL_NULL;
simsignal_t CanECU_Base::sendertempmessageIDSignal3 = SIMSIGNAL_NULL;

CanECU_Base::CanECU_Base()
{
        sendReqMsg = NULL;
}

CanECU_Base::~CanECU_Base()
{
        cancelAndDelete(sendReqMsg);
}

void CanECU_Base::initialize()
{
        UserInput = 2;
        Srv_initialize();
        Cli_initialize();
        isDoSs = false;
        TempID = false;
}

void CanECU_Base::Cli_initialize()
{
    packetsReceived = 0;
```

```
    rcvdPkSignal = registerSignal("rcvdPk");
    messageIDSignal = registerSignal("messageID");
    DataSignal = registerSignal("Data");
    sendermessageIDSignal = registerSignal("sendermessageID");
    sendertempmessageIDSignal1 = registerSignal("sendertempmessageID1");
    sendertempmessageIDSignal2 = registerSignal("sendertempmessageID2");
    sendertempmessageIDSignal3 = registerSignal("sendertempmessageID3");
    num = 0;

    WATCH(packetsReceived);
}


void CanECU_Base::handleMessage(cMessage *msg)
{

        if(msg->getKind() != 0){
        std::map<int,BasicMessageInfo>::iterator itr;
        itr = BasicMessageInfoList.find(msg->getKind());
        BasicMessageInfo value;
        value = itr->second;

                if(itr != BasicMessageInfoList.end()){
                        handleSrvMessage(msg);
                } else if(itr == BasicMessageInfoList.end()) {
                }

        } else {
                handleCliMessage(msg);
        }


}

void CanECU_Base::handleCliMessage(cMessage *msg)
{
        cMessage *copy = new cMessage;
        copy = msg;
        CanTraffic *can_traffic = check_and_cast<CanTraffic *>(copy);

        if(can_traffic->getType() == CAN_MESSAGE)
                receiveCanFrame(check_and_cast<CanFrame*>(msg));
        else
                receiveCanRemoteFrame(check_and_cast<CanRemoteFrame*>(msg));
}
```

```
void CanECU_Base::receiveCanFrame(CanFrame *msg)
{
        EV << "Received CAN Frame '" << msg->getName() << "'\n";
        EV << "msg->getCreationTime: " << msg->getCreationTime() << endl;
    packetsReceived++;
    cPacket *packet = PK(msg);

    uint8_t* temp = msg->getData();
    uint8_t com1[8], key[16], midres[2], id[8];
    int tem = 0, tem2 = 0, Mid = 0, index;
    std::map<int,BasicMessageInfo>::iterator itr;
    BasicMessageInfo value;

    for(index = 0; index < 16; index++)
                key[index] = index;

    for(index = 0; index < 2; index++)
                midres[index] = 0;

  if(!isDoSs){
    int temps = 0, temps2 = 0;
    tem |=temp[7];
    tem2 |=temp[6];
    tem |= (tem2<<8);

    TempSGWID = tem+0x01;

    midres[1] |= (TempSGWID&255);
    midres[0] |= ((TempSGWID>>8)&255);
    siphash(id, midres, 2, key);

    temps |=id[7];
    temps2 |=(id[6]&7);
    temps |=(temps2<<8);
    TempSGWID = temps%33;

    if(TempSGWID == msg->getMessageID() && temp[5] == 1){
        EV<<"DoS attack detected"<<endl;
        isDoSs = true;
    }
  }

  if(isDoSs && !TempID){
        int ha = 0, ha2 = 0;
```

```
    for(index = 0; index < 2; index++)
                midres[index] = 0;

    for(index = 0; index < 8; index++)
                com1[index] = 0;

for(itr = BasicMessageInfoList.begin(); itr != BasicMessageInfoList.end(); ++itr){
  tem = 0, tem2 = 0;
  value = itr->second;
  temp = msg->getData();
  tem |=temp[7];
  tem2 |=temp[6];
  tem |= (tem2<<8);

  Mid = (tem+value.ID)/10;

  midres[1] |= (Mid&255);
  midres[0] |= ((Mid>>8)&255);

  siphash(com1, midres, 2, key);
  ha |=com1[7];
  ha2 |=(com1[6]&7);
  ha |=(ha2<<8);
  TemperalID= ha%33;

  EV<<"TempID: "<<TemperalID<<" OriginalID: "<<value.ID<<endl;
}
      TempID = true;

}

  if (isDoSs && TempID && TempSGWID == msg->getMessageID() && temp[5] == 0)
  {
      isDoSs = false;
      TempID = false;
      num++;
  }

  if(msg->getMessageID() == PreviousID && !isDoSs)
  {
              handleSelfSendRequest(MyID);
  }

  emit(rcvdPkSignal, packet);
  emit(messageIDSignal,msg->getMessageID());
```

```
    delete msg;
}

void CanECU_Base::receiveCanRemoteFrame(CanRemoteFrame *msg)
{
        EV << "Received CAN Remote Frame '"<< msg->getCreationTime()<< "'\n";
        EV << "msg->getCreationTime: " << msg->getCreationTime() << endl;
        EV << "msg->MessageID: " << msg->getMessageID() << endl;

        if((msg->getControl_field()&(0x1<<6)) && BackUp != NULL &&
        msg->getMessageID() == BackUp->getMessageID()){
                handleSelfHashSendRequest(BackUp->getMessageID());
        }
        else if(!(msg->getControl_field()&(0x1<<6))){
                handleSelfSendRequest(msg->getMessageID());
        }

    delete msg;
}

void CanECU_Base::Srv_initialize()
{
    packetsSent = packetsReceived = 0;
    sendermessageIDSignal = registerSignal("sendermessageID");
    sentPkSignal = registerSignal("sentPk");
    rcvdPkSignal = registerSignal("rcvdPk");
        sendInterval = &par("sendInterval");
    drift = par("drift");
    simtime_t startTime = par("startTime");
    BackUp = NULL;

    WATCH(packetsSent);
    WATCH(packetsReceived);
    WATCH(drift);

    cXMLElementList messages = ((par("message").xmlValue())->getChildrenByTagName("SendMessage"));
    cXMLElement *message;

    cXMLElementList RemoteMessages = ((par("message").xmlValue())->getChildrenByTagName("RemoteMessage"));
    cXMLElement *Remotemessage;

    BasicMessageInfo bmInfo;

    for (int i = 0; i < (int)messages.size(); i++) {
```

```
message = messages[i];
SendMessageInfo smInfo;

smInfo.ID = ToDec(message->getAttribute("ID"));
    MyID = smInfo.ID;
    bmInfo.ID = ToDec(message->getAttribute("ID"));
smInfo.DLC = atoi(message->getAttribute("DLC"));
smInfo.SendInterval = atof(message->getAttribute("SendInterval")) / 1000;
smInfo.Offset = atof(message->getAttribute("Offset")) / 1000000;

EV << "ID : " << smInfo.ID << "DLC : " << smInfo.DLC << " SendInterval : " << smInfo.SendInterval;

const char *SendTimes = message->getAttribute("SendTime");
smInfo.SendTime = cStringTokenizer(SendTimes).asDoubleVector();
cStringTokenizer tokenizer(SendTimes);

numsendTime = 0;
const char *token;

while ((token = tokenizer.nextToken()) != NULL)      {
            numsendTime++;
    }

EV << " SendTime : ";
for (int j = 0; j < numsendTime; j++) {
    smInfo.SendTime[j] = smInfo.SendTime[j] / 1000;
    EV << smInfo.SendTime[j] << " ";
}
EV << endl;

SendMessageInfoList.insert(std::map<int,SendMessageInfo>::value_type(smInfo.ID,smInfo));
    BasicMessageInfoList.insert(std::map<int,BasicMessageInfo>::value_type(bmInfo.ID,bmInfo));

if (smInfo.SendInterval == 0) {
    sendReqMsg = new cMessage("SendRequest1", bmInfo.ID);
    for (int j = 0; j < numsendTime; j++) {
            EV << "time1:" << (1+drift) << endl;
            EV << "time2:" << startTime << endl;
            EV << "time3:" << (static_cast<double>(smInfo.SendTime[j])*(1+drift)) << endl;
            scheduleAt(startTime + (double)((smInfo.SendTime[j])*(1+drift)), sendReqMsg->dup());
    }
} else {
    sendReqMsg = new cMessage("SendRequest2", bmInfo.ID);
            scheduleAt(startTime+(smInfo.Offset)*(1+drift), sendReqMsg);
}
```

```
}

for (int i = 0; i < (int)RemoteMessages.size(); i++) {

Remotemessage = RemoteMessages[i];
RemoteMessageInfo rmInfo;

rmInfo.ID = ToDec(Remotemessage->getAttribute("ID"));
    bmInfo.ID = ToDec(Remotemessage->getAttribute("ID"));
    rmInfo.Control = (0|(0x1<<6));
    rmInfo.RTR = atoi(Remotemessage->getAttribute("RTR"));
rmInfo.SendInterval = atof(Remotemessage->getAttribute("SendInterval")) / 1000;
rmInfo.Offset = atof(Remotemessage->getAttribute("Offset")) / 1000000;
    rmInfo.isSent = false;
    bmInfo.isSent = false;

    EV << "ID : " << rmInfo.ID<<" RM SendInterval : " << rmInfo.SendInterval;

const char *SendTimes = Remotemessage->getAttribute("SendTime");
rmInfo.SendTime = cStringTokenizer(SendTimes).asDoubleVector();
cStringTokenizer tokenizer(SendTimes);

numsendTime = 0;
const char *token;
while ((token = tokenizer.nextToken()) != NULL)      {
            numsendTime++;
    }

EV << " SendTime : ";
for (int j = 0; j < numsendTime; j++) {
    rmInfo.SendTime[j] = rmInfo.SendTime[j] / 1000;
    EV << rmInfo.SendTime[j] << " ";
}
EV << endl;


RemoteMessageInfoList.insert(std::map<int,RemoteMessageInfo>::value_type(rmInfo.ID,rmInfo));

    std::pair< std::map<int, BasicMessageInfo>::iterator, bool > Result;
    Result = BasicMessageInfoList.insert(std::map<int,BasicMessageInfo>::value_type(bmInfo.ID,bmInfo));

    if(Result.second == false){
            BasicMessageInfoList.insert(std::map<int,BasicMessageInfo>::value_type(bmInfo.ID+1,bmInfo));
            EV << "Here\n";
```

```
        }

        if (rmInfo.SendInterval == 0 && Result.second != false) {
                sendReqMsg = new cMessage("SendRequest5", bmInfo.ID);
                for (int j = 0; j < numsendTime; j++) {
                        EV << "time1:" << (1+drift) << endl;
                        EV << "time2:" << startTime << endl;
                        EV << "time3:" << (static_cast<double>(rmInfo.SendTime[j])*(1+drift)) << endl;
                        scheduleAt(startTime + (double)((rmInfo.SendTime[j])*(1+drift)), sendReqMsg->dup());
                }
        } else if(rmInfo.SendInterval == 0 && Result.second == false){
                EV << "HereHEre"<<bmInfo.ID+1<<"\n";
                sendReqMsg = new cMessage("SendRequest5", bmInfo.ID+1);
                for (int j = 0; j < numsendTime; j++) {
                        EV << "time1:" << (1+drift) << endl;
                        EV << "time2:" << startTime << endl;
                        EV << "time3:" << (static_cast<double>(rmInfo.SendTime[j])*(1+drift)) << endl;
                        scheduleAt(startTime + (double)((rmInfo.SendTime[j])*(1+drift)), sendReqMsg->dup());
                }
        } else {
                sendReqMsg = new cMessage("SendRequest6", bmInfo.ID);
                scheduleAt(startTime+(rmInfo.Offset)*(1+drift), sendReqMsg);
        }
    }
   case_Sequence();

        WATCH(numsendTime);
        WATCH(sendIntervaldouble);
    WATCH(messageID);
}

void CanECU_Base::case_Sequence()
{
    cXMLElementList case_Sequence = ((par("Case_Sequence").xmlValue())->getChildrenByTagName("Case"));
    cXMLElementList case_Sequence2 = case_Sequence[UserInput]->getChildren();
    cXMLElement *case_id;
    int temp;

   for(int i =0; i < (int)case_Sequence2.size(); i++){
    case_id = case_Sequence2[i];

    temp = ToDec(case_id->getAttribute("ID"));
    if(MyID == temp)
        PreviousID = ToDec(case_id->getAttribute("PreviousECU_ID"));
   }
```

```
}

void CanECU_Base::handleSrvMessage(cMessage *msg)
{
        if (!msg->isSelfMessage()) {
                CanTraffic *can_traffic = check_and_cast<CanTraffic *>(msg);
                switch (can_traffic->getType()) {
                case CAN_MESSAGE:
                {
                        break;
                }
                case CAN_REMOTE:
                {
                        break;
                }
                default:
                        error("Message with unexpected message type %d", can_traffic->getType());
                }
        }

        else {
                EV << "ECU CanAppSrv: Self-message " << msg << " received\n";
                generateMessage(msg->getKind());
        }
        delete msg;
}

void CanECU_Base::generateMessage(int MessageID)
{

        std::map<int,BasicMessageInfo>::iterator itr;
        std::map<int,BasicMessageInfo>::iterator itr2;
        itr = BasicMessageInfoList.find(MessageID);
        BasicMessageInfo value;
        value = itr->second;

        if(itr != BasicMessageInfoList.end() && value.type == CAN_MESSAGE){
                handleSelfSendRequest(MessageID);
        } else if(itr == BasicMessageInfoList.end()) {
                EV << "MessageID " << MessageID <<"is not included in SendMessageInfoList." << endl;
        }

        if(itr != BasicMessageInfoList.end() && value.type == CAN_REMOTE){
                handleSelfRemoteRequest(MessageID);
```

```
        }

}

void CanECU_Base::handleSelfSendRequest(int MessageID)
{
        std::map<int,SendMessageInfo>::iterator itr;
        itr = SendMessageInfoList.find(MessageID);
        CanFrame *msg = new CanFrame("hoge");
        SendMessageInfo value;
        uint8_t test[8];

        for(int i = 0; i < 8; i++){
                        test[i] = (rand()%255)+1;
        }


        if (itr != SendMessageInfoList.end()) {
                value = itr->second;

                if(!isDoSs)
                 msg->setMessageID(value.ID);
                else
                 msg->setMessageID(TemperalID);

                msg->setData(test);
                msg->setFrameByteLength(value.DLC);
                msg->setType(0);
                msg->setIsSent(!(value.isSent));
                char msgName[32];
                sprintf(msgName,"0x%04x",msg->getMessageID());
                msg->setName(msgName);
                BackUp = msg->dup();
        } else {
                EV << "MessageID " << MessageID <<"is not included in SendMessageInfoList." << endl;
        }

        EV << "CanAppSrv: Generating CAN Message '" << msg->getName() <<"("<< msg->getMessageID() << ")'\n";
        sendMessage(msg);
        EV << "CanAppSrv: Sent CAN Message '" << msg->getName() << "'\n";

        if (value.SendInterval != 0) {
                sendReqMsg = new cMessage("SendRequest3", value.ID);
                scheduleAt(simTime() + (value.SendInterval)*(1+drift), sendReqMsg);
        }
```

```
}

void CanECU_Base::handleSelfHashSendRequest(int MessageID)
{

        std::map<int,SendMessageInfo>::iterator itr;
        itr = SendMessageInfoList.find(MessageID);
        CanFrame *msg = new CanFrame("hoge");
        SendMessageInfo value;
        uint8_t key[16], i[1] ={0,}, *test;

        for(int i = 0; i < 16; i++)
                key[i] = i;

        test = BackUp->getData();

        if (itr != SendMessageInfoList.end()) {

                value = itr->second;

                siphash(test, i, 1, key);
                msg->setMessageID(value.ID);
                msg->setData(BackUp->getData());
                msg->setFrameByteLength(value.DLC);
                msg->setType(0);
                msg->setIsSent(!(value.isSent));
                char msgName[32];
                sprintf(msgName,"0x%04x",msg->getMessageID());
                msg->setName(msgName);
        } else {
                EV << "MessageID " << MessageID <<"is not included in SendMessageInfoList." << endl;
        }

        EV << "CanAppSrv: Generating CAN Hash Message '" << msg->getName()
        <<"("<< msg->getMessageID() << ")'\n";
        sendMessage(msg);
        EV << "CanAppSrv: Sent CAN Hash Message '" << msg->getName() << "'\n";

        if (value.SendInterval != 0) {
                sendReqMsg = new cMessage("SendRequest3", value.ID);
                scheduleAt(simTime() + (value.SendInterval)*(1+drift), sendReqMsg);
        }

}
```

```
void CanECU_Base::handleSelfRemoteRequest(int MessageID)
{
        std::map<int,RemoteMessageInfo>::iterator Remoteitr;
        Remoteitr = RemoteMessageInfoList.find(MessageID);

        if(Remoteitr == RemoteMessageInfoList.end()){
                Remoteitr = RemoteMessageInfoList.find(MessageID-1);
                if(Remoteitr == RemoteMessageInfoList.end()){
                        EV<<"Remote Frame creation Error \n";
                }
        }


        CanRemoteFrame *Remotemsg = new CanRemoteFrame("hoge");
        RemoteMessageInfo Remotevalue;

        if (Remoteitr != RemoteMessageInfoList.end()) {
                Remotevalue = Remoteitr->second;
                Remotemsg->setMessageID(Remotevalue.ID);
                Remotemsg->setControl_field(Remotevalue.Control);
                Remotemsg->setType(5);
                Remotemsg->setIsSent(!(Remotevalue.isSent));
                char RemotemsgName[32];
                sprintf(RemotemsgName,"0x%04x",Remotemsg->getMessageID());
                Remotemsg->setName(RemotemsgName);
        } else {
                EV << "MessageID " << MessageID <<"is not included in SendMessageInfoList." << endl;
        }

        EV << "CanAppSrv: Generating CAN Remote Message '" << Remotemsg->getName()
        <<"("<< Remotemsg->getMessageID() << ")'\n";
        sendRemoteMessage(Remotemsg);
        EV << "CanAppSrv: Sent CAN Remote Message '" << Remotemsg->getName() << "'\n";

        if (Remotevalue.SendInterval != 0) {
                sendReqMsg = new cMessage("SendRequest4", Remotevalue.ID);
                scheduleAt(simTime() + (Remotevalue.SendInterval)*(1+drift), sendReqMsg);
        }
}

void CanECU_Base::handleCanMessage(cMessage *msg)
{

    cMessage *copy = new cMessage;
    copy = msg;
```

```
    CanFrame *can_traffic = check_and_cast<CanFrame *>(copy);

    if (can_traffic->getType() == CAN_MESSAGE) {
     EV << "CanAppSrv: Received packet '" << msg->getName() << "'\n";
     CanFrame *req = check_and_cast<CanFrame *>(msg);
     packetsReceived++;
     emit(rcvdPkSignal, req);

     long MessageID = req->getMessageID() + 1;
     uint8_t* Data;
     Data = req->getData();
     char msgname[30];
     strcpy(msgname, msg->getName());
     delete msg;
         EV << "CanAppSrv: Generating packet '" << msgname << "'\n";

         CanFrame *new_msg = new CanFrame(msgname);
         if(!isDoSs)
            new_msg->setMessageID(MessageID);
         else
            new_msg->setMessageID(TemperalID);
         new_msg->setData(Data);
         new_msg->setFrameByteLength(2);
         new_msg->setIsSent(true);
         BackUp = new_msg->dup();
         sendMessage(new_msg);
    } else {
     CanRemoteFrame *req = check_and_cast<CanRemoteFrame *>(msg);
     long MessageID = req->getMessageID() + 1;
     char Control = req->getControl_field();
     char Remotemsgname[30];
     strcpy(Remotemsgname, msg->getName());
     delete msg;

         CanRemoteFrame *new_Remotemsg = new CanRemoteFrame(Remotemsgname);
         new_Remotemsg->setMessageID(MessageID);
         new_Remotemsg->setControl_field(Control);
         new_Remotemsg->setFrameByteLength(2);
         new_Remotemsg->setIsSent(true);
         sendRemoteMessage(new_Remotemsg);
    }
}

void CanECU_Base::sendMessage(CanFrame* msg)
```

```
{
        emit(sentPkSignal, msg);
        if(isDoSs && num ==0)
        {
      emit(sendertempmessageIDSignal1,TemperalID);
        }else if(isDoSs && num ==1)
        {
          emit(sendertempmessageIDSignal2,TemperalID);
        }else if(isDoSs && num ==2)
        {
          emit(sendertempmessageIDSignal3,TemperalID);
        }
        else
        {
          emit(sendermessageIDSignal,msg->getMessageID());
        }
        send(msg, "out");
        packetsSent++;

}

void CanECU_Base::sendRemoteMessage(CanRemoteFrame *msg)
{
        send(msg, "out");
}

void CanECU_Base::finish()
{
}

unsigned long CanECU_Base::ToDec(const char str[])
{
   short i = 0;
   short n;
   unsigned long x = 0;
   char c;

   while (str[i] != '\0') {
       if ('0' <= str[i] && str[i] <= '9') {
           n = str[i] - '0';
           } else if ('a' <= (c = tolower(str[i])) && c <= 'f') {
           n = c - 'a' + 10;
           } else {
           printf("%s",str);
           printf("invalid charactor \n");
```

```
            exit(0);
        }
        i++;
        x = x * 16 + n;
    }
    return (x);
}
```

Attacker ECU Code

```
#include <stdio.h>
#include <string.h>
#include <math.h>

#include "Attacker_CanECU.h"
#include "CanApp_m.h"
#include "CanFrame_m.h"
#include "siphash.h"
#include <time.h>

Define_Module(Attacker_CanECU);

simsignal_t Attacker_CanECU::sentPkSignal = SIMSIGNAL_NULL;
simsignal_t Attacker_CanECU::rcvdPkSignal = SIMSIGNAL_NULL;
simsignal_t Attacker_CanECU::messageIDSignal = SIMSIGNAL_NULL;
simsignal_t Attacker_CanECU::DataSignal = SIMSIGNAL_NULL;
simsignal_t Attacker_CanECU::sendermessageIDSignal = SIMSIGNAL_NULL;

Attacker_CanECU::Attacker_CanECU()
{
        sendReqMsg = NULL;
}

Attacker_CanECU::~Attacker_CanECU()
{
        cancelAndDelete(sendReqMsg);
}

void Attacker_CanECU::initialize()
{
        srand((unsigned)time(NULL));
        Srv_initialize();
        Cli_initialize();
        isDoSs = false;
        TempID = false;
}
```

```
void Attacker_CanECU::Cli_initialize()
{
    packetsReceived = 0;
    rcvdPkSignal = registerSignal("rcvdPk");
    messageIDSignal = registerSignal("messageID");
    DataSignal = registerSignal("Data");
    sendermessageIDSignal = registerSignal("sendermessageID");

    WATCH(packetsReceived);
}


void Attacker_CanECU::handleMessage(cMessage *msg)
{

        if(msg->getKind() != 0){
        std::map<int,BasicMessageInfo>::iterator itr;
        itr = BasicMessageInfoList.find(msg->getKind());
        BasicMessageInfo value;
        value = itr->second;

                if(itr != BasicMessageInfoList.end()){
                        handleSrvMessage(msg);
                } else if(itr == BasicMessageInfoList.end()) {
                }

        } else {
                handleCliMessage(msg);
        }


}

void Attacker_CanECU::handleCliMessage(cMessage *msg)
{
        cMessage *copy = new cMessage;
        copy = msg;
        CanTraffic *can_traffic = check_and_cast<CanTraffic *>(copy);

        if(can_traffic->getType() == CAN_MESSAGE)
                receiveCanFrame(check_and_cast<CanFrame*>(msg));
        else
                receiveCanRemoteFrame(check_and_cast<CanRemoteFrame*>(msg));
}
```

```
void Attacker_CanECU::receiveCanFrame(CanFrame *msg)
{
        EV << "Received CAN Frame '" << msg->getName() << "'\n";
        EV << "msg->getCreationTime: " << msg->getCreationTime() << endl;
    packetsReceived++;
    cPacket *packet = PK(msg);

    emit(rcvdPkSignal, packet);
    emit(messageIDSignal,msg->getMessageID());
    delete msg;
}

void Attacker_CanECU::receiveCanRemoteFrame(CanRemoteFrame *msg)
{
        EV << "Received CAN Remote Frame '"<< msg->getCreationTime()<< "'\n";
        EV << "msg->getCreationTime: " << msg->getCreationTime() << endl;
        EV << "msg->MessageID: " << msg->getMessageID() << endl;
                if(MyID == msg->getMessageID())
                        handleSelfSendRequest(msg->getMessageID());

    delete msg;
}

void Attacker_CanECU::Srv_initialize()
{
    packetsSent = packetsReceived = 0;
    sendermessageIDSignal = registerSignal("sendermessageID");
    sentPkSignal = registerSignal("sentPk");
    rcvdPkSignal = registerSignal("rcvdPk");
        sendInterval = &par("sendInterval");
    drift = par("drift");
    simtime_t startTime = par("startTime");
    BackUp = NULL;

    WATCH(packetsSent);
    WATCH(packetsReceived);
    WATCH(drift);

    cXMLElementList messages = ((par("message").xmlValue())->getChildrenByTagName("SendMessage"));
    cXMLElement *message;

    cXMLElementList RemoteMessages = ((par("message").xmlValue())->getChildrenByTagName("RemoteMessage"));
    cXMLElement *Remotemessage;
```

```
BasicMessageInfo bmInfo;

for (int i = 0; i < (int)messages.size(); i++) {
    message = messages[i];
    SendMessageInfo smInfo;
    smInfo.ID = ToDec(message->getAttribute("ID"));
    MyID = smInfo.ID;
    bmInfo.ID = ToDec(message->getAttribute("ID"));
    smInfo.DLC = atoi(message->getAttribute("DLC"));
    smInfo.SendInterval = atof(message->getAttribute("SendInterval")) / 1000;
    smInfo.Offset = atof(message->getAttribute("Offset")) / 1000000;
    EV << "ID : " << smInfo.ID << "DLC : " << smInfo.DLC << " SendInterval : " << smInfo.SendInterval;

    const char *SendTimes = message->getAttribute("SendTime");
    smInfo.SendTime = cStringTokenizer(SendTimes).asDoubleVector();
    cStringTokenizer tokenizer(SendTimes);

    numsendTime = 0;
    const char *token;
    while ((token = tokenizer.nextToken()) != NULL) {
                numsendTime++;
        }

    EV << " SendTime : ";
    for (int j = 0; j < numsendTime; j++) {
        smInfo.SendTime[j] = smInfo.SendTime[j] / 1000;
        EV << smInfo.SendTime[j] << " ";
    }
    EV << endl;

SendMessageInfoList.insert(std::map<int,SendMessageInfo>::value_type(smInfo.ID,smInfo));
    BasicMessageInfoList.insert(std::map<int,BasicMessageInfo>::value_type(bmInfo.ID,bmInfo));

    if (smInfo.SendInterval == 0) {
        sendReqMsg = new cMessage("SendRequest1", bmInfo.ID);
        for (int j = 0; j < numsendTime; j++) {
            EV << "time1:" << (1+drift) << endl;
            EV << "time2:" << startTime << endl;
            EV << "time3:" << (static_cast<double>(smInfo.SendTime[j])*(1+drift)) << endl;
            scheduleAt(startTime + (double)((smInfo.SendTime[j])*(1+drift)), sendReqMsg->dup());
        }
    } else {
        sendReqMsg = new cMessage("SendRequest2", bmInfo.ID);
            scheduleAt(startTime+(smInfo.Offset)*(1+drift), sendReqMsg);
    }
```

```
}

for (int i = 0; i < (int)RemoteMessages.size(); i++) {
    Remotemessage = RemoteMessages[i];

RemoteMessageInfo rmInfo;
rmInfo.ID = ToDec(Remotemessage->getAttribute("ID"));
    bmInfo.ID = ToDec(Remotemessage->getAttribute("ID"));
    rmInfo.Control = (0|(0x1<<6));
    rmInfo.RTR = atoi(Remotemessage->getAttribute("RTR"));
rmInfo.SendInterval = atof(Remotemessage->getAttribute("SendInterval")) / 1000;
rmInfo.Offset = atof(Remotemessage->getAttribute("Offset")) / 1000000;
    rmInfo.isSent = false;
    bmInfo.isSent = false;

    EV << "ID : " << rmInfo.ID<<" RM SendInterval : " << rmInfo.SendInterval;



    const char *SendTimes = Remotemessage->getAttribute("SendTime");
    rmInfo.SendTime = cStringTokenizer(SendTimes).asDoubleVector();
    cStringTokenizer tokenizer(SendTimes);

    numsendTime = 0;
    const char *token;
    while ((token = tokenizer.nextToken()) != NULL) {
                numsendTime++;
        }

    EV << " SendTime : ";
    for (int j = 0; j < numsendTime; j++) {
        rmInfo.SendTime[j] = rmInfo.SendTime[j] / 1000;
        EV << rmInfo.SendTime[j] << " ";
    }
    EV << endl;

    RemoteMessageInfoList.insert(std::map<int,RemoteMessageInfo>::value_type(rmInfo.ID,rmInfo));

    std::pair< std::map<int, BasicMessageInfo>::iterator, bool > Result;
    Result = BasicMessageInfoList.insert(std::map<int,BasicMessageInfo>::value_type(bmInfo.ID,bmInfo));

    if(Result.second == false){
        BasicMessageInfoList.insert(std::map<int,BasicMessageInfo>::value_type(bmInfo.ID+1,bmInfo));
    }

    if (rmInfo.SendInterval == 0 && Result.second != false) {
```

```
                sendReqMsg = new cMessage("SendRequest5", bmInfo.ID);
                for (int j = 0; j < numsendTime; j++) {
                        EV << "time1:" << (1+drift) << endl;
                        EV << "time2:" << startTime << endl;
                        EV << "time3:" << (static_cast<double>(rmInfo.SendTime[j])*(1+drift)) << endl;
                        scheduleAt(startTime + (double)((rmInfo.SendTime[j])*(1+drift)), sendReqMsg->dup());
                }
        } else if(rmInfo.SendInterval == 0 && Result.second == false){
                EV << "HereHEre"<<bmInfo.ID+1<<"\n";
                sendReqMsg = new cMessage("SendRequest5", bmInfo.ID+1);
                for (int j = 0; j < numsendTime; j++) {
                        EV << "time1:" << (1+drift) << endl;
                        EV << "time2:" << startTime << endl;
                        EV << "time3:" << (static_cast<double>(rmInfo.SendTime[j])*(1+drift)) << endl;
                        scheduleAt(startTime + (double)((rmInfo.SendTime[j])*(1+drift)), sendReqMsg->dup());
                }
        } else {
                sendReqMsg = new cMessage("SendRequest6", bmInfo.ID);
                        scheduleAt(startTime+(rmInfo.Offset)*(1+drift), sendReqMsg);
        }
    }

        WATCH(numsendTime);
        WATCH(sendIntervaldouble);
        WATCH(messageID);
}

void Attacker_CanECU::handleSrvMessage(cMessage *msg)
{
        if (!msg->isSelfMessage()) {
                CanTraffic *can_traffic = check_and_cast<CanTraffic *>(msg);
                switch (can_traffic->getType()) {
                case CAN_MESSAGE:
                {
                        break;
                }
                case CAN_REMOTE:
                {
                        break;
                }
                default:
                        error("Message with unexpected message type %d", can_traffic->getType());
                }
        }
```

```
        else {

                EV << "ECU_CanAppSrv:_Self-message_" << msg << "_received\n";
                generateMessage(msg->getKind());
        }
        delete msg;
}

void Attacker_CanECU::generateMessage(int MessageID)
{

        std::map<int,BasicMessageInfo>::iterator itr;
        std::map<int,BasicMessageInfo>::iterator itr2;
        itr = BasicMessageInfoList.find(MessageID);
        BasicMessageInfo value;
        value = itr->second;


        if(itr != BasicMessageInfoList.end() && value.type == CAN_MESSAGE){
                handleSelfSendRequest(MessageID);
        } else if(itr == BasicMessageInfoList.end()) {
                EV << "MessageID_" << MessageID <<"is_not_included_in_SendMessageInfoList." << endl;
        }

        if(itr != BasicMessageInfoList.end() && value.type == CAN_REMOTE){
                handleSelfRemoteRequest(MessageID);
        }

}

void Attacker_CanECU::handleSelfSendRequest(int MessageID)
{
        std::map<int,SendMessageInfo>::iterator itr;
        itr = SendMessageInfoList.find(MessageID);
        CanFrame *msg = new CanFrame("hoge");
        SendMessageInfo value;
        uint8_t test[8] = {0,1,3,4,5,6,7};
        int ID[10] = {11, 21, 31, 41, 51, 61, 71, 91, 101};
        int IDs = rand()%9+1;


        if (itr != SendMessageInfoList.end()) {

                value = itr->second;
```

```
                msg->setMessageID(ID[IDs]);
                msg->setData(test);
                msg->setFrameByteLength(4);
                msg->setType(0);
                msg->setIsSent(!(value.isSent));
                char msgName[32];
                sprintf(msgName,"0x%04x",msg->getMessageID());
                msg->setName(msgName);
                BackUp = msg->dup();
        } else {
                EV << "MessageID " << MessageID <<"is not included in SendMessageInfoList." << endl;
        }

        EV << "CanAppSrv: Generating CAN Message '" << msg->getName() <<"("<< msg->getMessageID() << ")'\n";
        sendMessage(msg);
        EV << "CanAppSrv: Sent CAN Message '" << msg->getName() << "'\n";

        if (value.SendInterval != 0) {
                sendReqMsg = new cMessage("SendRequest3", value.ID);
                scheduleAt(simTime() + (value.SendInterval)*(1+drift), sendReqMsg);
        }

        if(isDoSs)
          emit(sendermessageIDSignal,TemperalID);

}

void Attacker_CanECU::handleSelfHashSendRequest(int MessageID)
{
        std::map<int,SendMessageInfo>::iterator itr;
        itr = SendMessageInfoList.find(MessageID);
        CanFrame *msg = new CanFrame("hoge");
        SendMessageInfo value;
        uint8_t test[8], key[16], i[1] ={0,};

        for(int i = 0; i < 16; i++)
                key[i] = i;

        if (itr != SendMessageInfoList.end()) {
                value = itr->second;
                msg->setMessageID(value.ID);
                msg->setData(test);
                msg->setFrameByteLength(value.DLC);
                msg->setType(0);
                msg->setIsSent(!(value.isSent));
```

```
                char msgName[32];
                sprintf(msgName,"0x%04x",msg->getMessageID());
                msg->setName(msgName);
                BackUp = msg->dup();
        } else {
                EV << "MessageID " << MessageID <<"is not included in SendMessageInfoList." << endl;
        }

        EV << "CanAppSrv: Generating CAN Hash Message '"
        << msg->getName() <<"("<< msg->getMessageID() << ")'\n";
        sendMessage(msg);
        EV << "CanAppSrv: Sent CAN Hash Message '" << msg->getName() << "'\n";

        if (value.SendInterval != 0) {
                sendReqMsg = new cMessage("SendRequest3", value.ID);
                scheduleAt(simTime() + (value.SendInterval)*(1+drift), sendReqMsg);
        }

}

void Attacker_CanECU::handleSelfRemoteRequest(int MessageID)
{
        std::map<int,RemoteMessageInfo>::iterator Remoteitr;
        Remoteitr = RemoteMessageInfoList.find(MessageID);

        if(Remoteitr == RemoteMessageInfoList.end()){
                Remoteitr = RemoteMessageInfoList.find(MessageID-1);
                if(Remoteitr == RemoteMessageInfoList.end()){
                        EV<<"Remote Frame creation Error \n";
                }
        }


        CanRemoteFrame *Remotemsg = new CanRemoteFrame("hoge");
        RemoteMessageInfo Remotevalue;



        if (Remoteitr != RemoteMessageInfoList.end()) {
                Remotevalue = Remoteitr->second;
                Remotemsg->setMessageID(Remotevalue.ID);
                Remotemsg->setControl_field(Remotevalue.Control);
                Remotemsg->setType(5);
                Remotemsg->setIsSent(!(Remotevalue.isSent));
                char RemotemsgName[32];
```

```
            sprintf(RemotemsgName,"0x%04x",Remotemsg->getMessageID());
            Remotemsg->setName(RemotemsgName);
        } else {
            EV << "MessageID " << MessageID <<"is not included in SendMessageInfoList." << endl;
        }

        EV << "CanAppSrv: Generating CAN Remote Message '"
        << Remotemsg->getName() <<"("<< Remotemsg->getMessageID() << ")'\n";
        sendRemoteMessage(Remotemsg);
        EV << "CanAppSrv: Sent CAN Remote Message '" << Remotemsg->getName() << "'\n";

        if (Remotevalue.SendInterval != 0) {
            sendReqMsg = new cMessage("SendRequest4", Remotevalue.ID);
            scheduleAt(simTime() + (Remotevalue.SendInterval)*(1+drift), sendReqMsg);
        }
}

void Attacker_CanECU::handleCanMessage(cMessage *msg)
{

    cMessage *copy = new cMessage;
    copy = msg;

    CanFrame *can_traffic = check_and_cast<CanFrame *>(copy);

    if (can_traffic->getType() == CAN_MESSAGE) {
     EV << "CanAppSrv: Received packet '" << msg->getName() << "'\n";
     CanFrame *req = check_and_cast<CanFrame *>(msg);
     packetsReceived++;
     emit(rcvdPkSignal, req);

     long MessageID = req->getMessageID() + 1;
     uint8_t* Data;
     Data = req->getData();
     char msgname[30];
     strcpy(msgname, msg->getName());
     delete msg;
        EV << "CanAppSrv: Generating packet '" << msgname << "'\n";

        CanFrame *new_msg = new CanFrame(msgname);
            new_msg->setMessageID(MessageID);
        new_msg->setData(Data);
        new_msg->setFrameByteLength(2);
        new_msg->setIsSent(true);
        BackUp = new_msg->dup();
```

```
        sendMessage(new_msg);
    } else {
     CanRemoteFrame *req = check_and_cast<CanRemoteFrame *>(msg);
     long MessageID = req->getMessageID() + 1;
     char Control = req->getControl_field();
     char Remotemsgname[30];
     strcpy(Remotemsgname, msg->getName());
     delete msg;

        CanRemoteFrame *new_Remotemsg = new CanRemoteFrame(Remotemsgname);
        new_Remotemsg->setMessageID(MessageID);
        new_Remotemsg->setControl_field(Control);
        new_Remotemsg->setFrameByteLength(2);
        new_Remotemsg->setIsSent(true);
        sendRemoteMessage(new_Remotemsg);
  }

  if(isDoSs)
          emit(sendermessageIDSignal,TemperalID);

}

void Attacker_CanECU::sendMessage(CanFrame* msg)
{
        emit(sentPkSignal, msg);
        emit(sendermessageIDSignal,msg->getMessageID());
        send(msg, "out");
        packetsSent++;

}

void Attacker_CanECU::sendRemoteMessage(CanRemoteFrame *msg)
{
        send(msg, "out");
}

void Attacker_CanECU::finish()
{
}

unsigned long Attacker_CanECU::ToDec(const char str[])
{
   short i = 0;
   short n;
   unsigned long x = 0;
```

```
    char c;

    while (str[i] != '\0') {
        if ('0' <= str[i] && str[i] <= '9') {
            n = str[i] - '0';
            } else if ('a' <= (c = tolower(str[i])) && c <= 'f') {
            n = c - 'a' + 10;
            } else {
            printf("%s",str);
            printf("invalid charactor \n");
            exit(0);
        }
        i++;
        x = x * 16 + n;
    }
    return (x);
}
```

Summary

A Countermeasure against Spoofing and DoS Attacks based on Message Sequence and Temporary ID in CAN

ICT의 발전은 빅 데이터, 모바일, 웨어러블 등 다양한 분야의 영향을 주었다. 자동차 분야 역시 ICT의 영향을 받아 자동차의 컨트롤을 해주는 ECU가 등장하게 되었다. 각 ECU들이 혼자서 동작하지 않고 데이터를 주고 받아야 하기 때문에, ECU들 끼리 통신 할 수 있는 네트워크가 필요하게 되었고 그 결과, CAN, LIN, FlexRay 같은 자동차 전용 네트워크 프로토콜이 도입되게 되었다. 이로 인해, ECU들은 효율적으로 데이터를 다른 ECU들에게 전송할 수 있으며, 이는 자동차의 컨트롤을 더욱 효율적으로 만들었다.

CAN, LIN, FlexRay 중에서 CAN은 현재 자동차용 네트워크 표준으로 거의 모든 자동차가 CAN을 자동차 네트워크 프로토콜로 사용중이다. 하지만 비록 CAN이 가장 널리 사용되는 자동차용 네트워크 프로토콜 이지만, CAN이 가지고 있는 브로드 캐스트 환경, 중재기능 같은 특징으로 인해 CAN에 대한 보안은 매우 취약한 상황이다. 이러한 취약한 특징들을 이용하여, 스푸핑 및 DoS 공격이 CAN에서 쉽게 이루어진다. 이러한 CAN의 취약점을 해결하기 위해, IDS, MAC, 암호 알고리즘 중 하나인 AES를 사용하는 등 수많은 아이디어들이 제시되었다. 그러나 제시된 아이디어들은 트래픽 증가, 기존 시스템에 미치는 영향, 도입 비용 등에서 문제를 가지고 있다. 또한, 몇몇 아이디어들은 자동차가 가지고 있는 특징으로 인해 검증이 되지 않아 아이디어의 효율성이나 효과를 증명하지 못하고 있는 상태이다.

본 논문에서는 기존 CAN에서 사용되는 게이트웨이를 변형한 보안 게이트웨이를 이용하여 스푸핑 및 DoS 공격을 방어할 수 있는 방법을 제시한다. 스푸핑 공격에 경우, 운전자의 행위에 기반한 메지시 순서를 이용하여 방어를 한다. 메시지의 순서를 저장한 테이블을 만들고 이를 이용하여 스푸핑 공격을 탐지한다. 그리고 별도의 SipHash를 이용한 검증 과정을 이용하여 메시지의 유효한지 아닌지를 결정하여 방어를 한다. 또한 시드값과 SipHash를 이용한 임시 id를 사용하여 DoS 공격을 방어한다.

제시된 아이디어의 검증을 위해, 네트워크 시뮬레이터 중 하나인 OMNeT++가 사용되었다. 제시된 아이디어는 높은 탐지율과 낮은 트래픽 증가량을 보여준다. 또한, DoS 공격에 경우, 제시된 아이디어는 DoS 공격이 아무런 효과가 없음을 프레임 손실률을 통해 보여준다.

핵심어: CAN, IDS, Security gateway, spoofing attack, DoS attack.

감 사 의 글

이 논문을 완성하기까지 주위의 모든 분들로부터 수많은 도움을 받았습니다. 지도교수님이신 김광조 교수님께는 연구의 기본부터 시작하여 연구자로써 가져야 할 자세까지 많은 부분을 배울 수 있었고 이는 제가 앞으로 연구자로써 살아가는데 큰 도움이 되었습니다. 또한, 바쁜 시간을 내어 주시어 학위논문심사에 참여해주시고 좋은 조언을 해주신 하정석 교수님, 신승원 교수님께도 감사의 말씀을 드립니다.

그리고 암호와 정보보호 연구실에서 함께 연구실 생활을 한 선후배님(박준정 형님, 정제성 형님, 김학주, 최락용, 김경민, 홍진아, Aminanto Erza Muhamad)들에게 연구실 생활 및 연구 등 많은 부분에서 도움을 받았고 이에 감사의 말을 전합니다.

또한, 함께 수업을 들으며, 이야기를 나누었던 정보보호대학원 7기 동기(박준정 형님, 정제성 형님, 김정민, 배찬우, 이승수, 강희도, 김연근, 서상아, 박수완, 김은수)들에게도 감사를 드립니다. 특히 저에게 큰 도움을 준 박준정 형님, 김정민, 배찬우에게 깊은 감사의 말을 전합니다.

끝으로 오늘의 제가 있을 수 있도록 항상 진심어린 사랑으로 키워 주신 어머니와 하늘에 계신 아버지, 수영이형에게 깊은 감사드립니다.

이 력 서

이 름 : 안 수 현

생 년 월 일 : 1987년 10월 15일

E-mail 주 소 : ahn1015@kaist.ac.kr

학 력

2003. 3. – 2006. 2. 동화고등학교

2007. 3. – 2014. 2. 숭실대학교 정보통신전자공학부 (B.S.)

2014. 3. – 2016. 2. 한국과학기술원 정보보호대학원 (M.S.)

연 구 과 제

2014. 7. – 2015. 6. Intrusion Detection System for Infrastructures Using Big Data Analytics

2015. 4. – 2015. 11. Research on the Security Analysis of Post Quantum Crytosystem

2015. 7. – 2016. 3. Authenticated Security for Smart-Grid Using Big-Data Analytics

2015. 3. – 2016. 2. 생체모방 알고리즘(Bio-inspired Algorithm)을 활용한 통신기술 연구

연 구 업 적

1. **안수현**, 김광조, "IoT 환경에 적합한 경량 DTLS 프로토콜 구성 방법", 한국정보보호학회 동계학술대회 (CISC-W'14), 2014.12.06. 한양대학교, 서울
2. **안수현**, 김광조, "Arduino를 이용한 CAN의 소규모 실험 환경 구축 및 보안 취약성 검증", 한국정보보호학회 하계학술대회(CISC-S'15), 2015.06.25-26. 한국과학기술원, 대전
3. **안수현**, 김광조, "CAN에서 보안 게이트웨이를 이용한 침입탐지기법제안", 2015 정보보호학술발표회 논문집 충청지부, pp.100-103, 서원대학교, 청주.

4. **안수현**, 김광조, "CAN에서 보안 게이트웨이를 이용한 침입탐지기법", 한국정보보호학회 동계학술대회 (CISC-W'15), 2015.12.05. 서울여자대학교, 서울

5. 배찬우, **안수현**, 박준정, 신승원, 김광조, "웹 스캐닝을 이용한 캠퍼스 내 웹사이트 보안 취약점 발굴 및 웹 보안 연구", 2015 정보보호학술발표회논문집 충청지부, pp.42-45, 서원대학교, 청주.

6. 박준정, 김소라, **안수현**, 임채호, 김광조, "조직의 실시간 보안관리 체계 확립을 위한 '인터페이스보안' 강화에 대한 연구", 정보처리학회지 제4권 제5호 pp.171 - 177, 2015.5

7. 정제성, 김경민, 김학주, 박준정, **안수현**, 이동수, 최락용, 김광조, 김대영,"경량 암호를 이용한 IoT Secure SNAIL 플랫폼 구성(Ⅰ)", 한국정보보호학회 동계학술대회(CISC-W'14), 2014.12.06. 한양대학교, 서울