Journal of Information Processing Vol.25 220–228 (Feb. 2017)

[DOI: 10.2197/ipsjjip.25.220]

Regular Paper

Automotive Attacks and Countermeasures on LIN-Bus

Junko Takahashi[1,a)] Yosuke Aragane[1] Toshiyuki Miyazawa[2] Hitoshi Fuji[1]
Hirofumi Yamashita[3,b)] Keita Hayakawa[3] Shintarou Ukai[3] Hiroshi Hayakawa[3]

Received: May 26, 2016, Accepted: November 1, 2016

Abstract: This paper presents a security analysis of the Local Interconnect Network (LIN) that is used in assembly units such as seats, steering wheels, and doors in vehicles. Recently, the number of security threats to in-vehicle networks such as the Controller Area Network has increased. In contrast, there have been no reports that evaluate the security of LIN in detail. The security analysis of LIN is important because it is used in units related to seats, steering wheels, etc. and it is at risk for an attack. In this paper, we present the first evaluation on the security of LIN. We present case studies of attacks that use the characteristics of a commonly-used error handling mechanism. In the attacks, the attacker intentionally stops communication using the error handling mechanism and sends a false response in place of a valid one. We experimentally show the feasibility of the attacks using a vehicle microcontroller. Furthermore, we present countermeasures against the attacks. The results of this study show that there is vulnerability to attack when the error handling mechanism is simply designed. We believe that this study will contribute to improvements in security of in-vehicle communications.

Keywords: vehicle security, in-vehicle protocol, LIN, error handling mechanism

1. Introduction

Modern vehicles already contain a number of microcontrollers that are responsible for crucial components such as the control of the engine, brakes, and steering wheel. Recent vehicle technologies such as automated driving assistance have been achieved through advances in microcontrollers. Each microcontroller in a vehicle communicates through an in-vehicle network such as the Controller Area Network (CAN), Local Interconnect Network (LIN), or FlexRay. These networks are applied to vehicle functions related to, for example, the engine, body, or multimedia control based on their essential technical properties and application areas.

Recently, some investigations have highlighted security risks to in-vehicle communications such as CAN because they are not designed considering security [1], [2], [3], [4], [5]. These studies showed that any arbitrary action could be induced by sending a false message to the CAN bus. Some detection and prevention mechanisms such as message authentication code (MAC) on CAN were also proposed in Refs. [6], [7], [8], [9] and specifications for the MAC on the bus were developed by the automotive open system architecture (AUTOSAR), which is a worldwide development partnership of vehicles (specification version is 4.2.2) [10].

On the other hand, if an attacker can intentionally control the communications on LIN, which is used in steering wheels and doors [11], there is the possibility for a significant threat. For example, while the vehicle is moving on the highway, the user could be exposed to risk if an attacker maliciously tampered with the controls related to the steering wheel or doors. Although the possibility of an attack on LIN is briefly described in a previous study [3], detailed attack methods and the countermeasures were not shown. Furthermore, LIN uses the master-slave method which is different from that of CAN. Therefore, it is difficult to simply apply the CAN attack method to LIN.

In this paper, we evaluate the resistance of LIN to an attack that leads to malicious control of a vehicle[*1]. We present case studies of attacks, in which the attacker can send a false message with the receiver identifying it a valid one using a general error handling mechanism [13], [14].

Although LIN is commonly used in a CAN subnetwork, recent hacking techniques [1], [15] could make it easier to gain accesses to the LIN bus through the CAN-bus node. We also perform experimental analysis of the proposed attacks using a vehicle microcontroller in which the LIN protocol is implemented. Furthermore, we propose countermeasures for the attack.

The remainder of this paper is organized as follows. An overview of LIN is given and previous work is described in Sections 2 and 3, respectively. We describe the concept behind the

[1] NTT Secure Platform Laboratories, Musashino, Tokyo 180–8585, Japan
[2] NIPPON TELEGRAPH AND TELEPHONE CORPORATION, Chiyoda, Tokyo 100–8116, Japan
[3] DENSO CORPORATION, Kariya, Aichi 448–8661, Japan
[a)] takahashi.junko@lab.ntt.co.jp
[b)] HIROHUMI_H_YAMASHITA@denso.co.jp

[*1] A preliminary version of this paper was published in the conference proceedings of SCIS 2016 [12]. In this paper, we change the constructions of sections and delete the redundant representations for reader-friendly compared to Ref. [12]. Furthermore, we reveal that the attack regarding the header collisions described in Section 4.2.3 of Ref. [12] is not effective. Thus, we correct this attack method and the successful attack is described in Section 5.3 in this paper. Corresponding to the above, we delete the countermeasure regarding the header resending in Section 6 (3) of Ref. [12] and we describe a countermeasure in Section 7.3 which is devised after the SCIS conference.

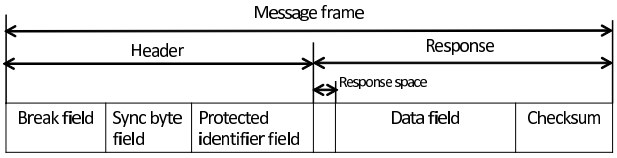

Fig. 1 LIN message frame.

study in Section 4 and the case studies of the attacks on LIN in Section 5. We present experimental results of the proposed attacks in Section 6 and propose countermeasures in Section 7. Finally, we conclude the paper in Section 8.

2. LIN Specifications

This section gives an overview of LIN [16].

2.1 Introduction to LIN

LIN is developed by the LIN consortium to create a standard for low cost and low-end multiplexed communications in automotive networks. Typical applications for LIN include assembly units such as seats, doors, engines, and steering wheels [11] with typical data rates of up to 20 kbps. LIN is based on a master/slave architecture that comprises a single master node and numerous slave nodes connected using a single wire. LIN uses a time trigger method that employs a priori fixed time schedule, which is the key property of LIN. The master node has the time schedule that decides when and which message frame shall be transmitted in the bus.

2.2 Message Frame

Message frames consist of a header provided by the master node and a response provided by the master/slave node as shown in **Fig. 1**. The header consists of a break field that is used to signal the beginning of a new frame. The sync byte field is a field with the data value of `0x55` that is used by the slave node for clock synchronization. The protected identifier (PID) field of the header consists of two sub-fields: the frame identifier (bits 0 to 5) and the parity (bits 6 and 7). The PID denotes the contents of a specific message but not the destination. The data field of the response stores up to 8 bytes of data. The checksum is used for verification so that the node receives data correctly.

2.3 Transmission Method

Several frame types for transmission are used in LIN. Here, we describe the unconditional frame that is used in this investigation. The unconditional frame is the most commonly used and it carries data. An example transmission method is shown in **Fig. 2**. In the example, slave node 2 performs a task based on the response sent from slave node 1. At this time, LIN uses the master-slave method. Using this method, slave node 1 can only send a response based on the header send from the master node. The timing of the header is controlled by the time schedule of the master node. At the beginning, the master node sends a header including the PID (`0x00` in this case) which indicates the sender or the receiver as indicated by (A) in Fig. 2. The header is broadcast in the bus and all nodes receive it. The node corresponding to the PID sends a response or receives it (in this case, slave node 1 sends the response and slave node 2 receives the response) as indicated by (B) in Fig. 2. A response without the header will be simply dropped in the transmission of LIN.

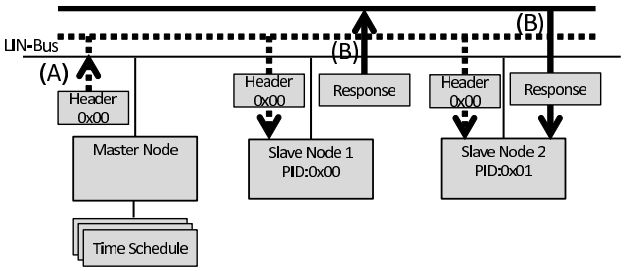

Fig. 2 LIN transmission method.

3. Related Work

In this section, we describe related work. Because the in-vehicle protocols are not designed considering security, it is possible to implement some attacks such as spoofing and denial-of-service (DoS) attacks that are famous in the IT field. In previous studies, most papers focus on the CAN-bus security, and attack methods were proposed in Refs. [1], [2], [4], [5]. In Refs. [1], [4], hacking techniques to inject a false message into the CAN bus using the diagnostic interface of the vehicle were introduced and they showed that an arbitrary action could be performed. Hoppe, Kiltz, and Dittmann [2] showed several automotive system vulnerabilities such as in the window lift and airbag control using real automotive hardware and presented short-term countermeasures. They also showed basic attack principles from the aspect of automotive IT security [5].

Regarding the investigation of the security of other in-vehicle networks, the security of FlexRay was studied in Ref. [17]. The security on the LIN bus was briefly introduced in Ref. [3]. In the attack on LIN, the authors described the attack possibility where each slave node would be completely deactivated by introducing well-directed false sleep frames. The slave nodes still remains the sleep mode until a wake-up mode is requested from the master node. They also described that modifying the sync byte field to any value is another attack point.

4. Concept Behind Study

In this section, we describe the difficulty of the attacks against LIN and the concept of the proposed attacks.

4.1 Difficulty of Attacks Against LIN

In the LIN-bus, in order to implement some malicious behavior, we must be able to (1) inject a false response in which a part of data is changed to any value into the bus depending on the kinds of data and (2) do so at any timing.

Regarding (1), to validate such a false response, we need to inject it after a header is sent from the master node. However, it is difficult to change a part of data of the false response by simply injecting it in the bus because it may collide with the correct response that is sent after the header.

Regarding (2), to control the timing of the sequence of the message which causes not to do normal operations, it is difficult to manipulate the time schedule because we need to physically ac-

cess the master node and tamper with it based on the details of the implementation method of the master node which we cannot obtain in general.

4.2 Concept Behind Proposed Attacks

In order to inject any value of a false response at any time, we employ the characteristics of the error handling mechanism. In this mechanism, the slave node (sender) stops sending a response when the data in the bus are different from the sent data in the response while monitoring the bus level and waits the next header. Such kind of errors is known as bit errors. This type of error handling mechanism is commonly used and implemented in the LIN protocol [13], [14]. In fact, we intentionally generate a collision between the responses to induce the bit error and inject a false response after an error occurs. Thus, we can send any value of the false response after the collisions occur because the sender stops sending the correct response and the receiver accepts the false response as a valid one. Although sending a false response as a valid one is sufficient to warrant risk, we also consider attacks in which the sequence of the transmission of the response is manipulated by generating the header collisions without tampering with the time schedule of the master node.

5. Evaluations of Attacks on LIN

In this section, we describe the assumptions in the study and the proposed attacks that lead to the malicious behavior.

5.1 Study Assumptions

Here, we describe the assumptions in the study.

- The sender slave node monitors the LIN bus in terms of bytes to determine whether the sent response is equal to the value in the bus. If they do not match, the slave node detects the bit error and stops sending the response [13], [14].
- An unconditional frame is used in the LIN transmission.
- The attacker can inject false headers/responses into the LIN bus through physical or logical access to the LIN bus while each node normally functions based on the time schedule. As an example, since the master node is usually connected to CAN [18], it is possible for the attacker to gain accesses to LIN through the master node using recent hacking techniques [1], [15].
- The attacker can guess the sequence of the message in the time schedule including the message frame that the attacker targets.
- The attacker can read and interpret the PID of the headers and the responses transmitted in the targeted LIN. It is possible to intercept them because the transmission of LIN is not encrypted. The attacker can guess the contents of the headers and the responses through analysis based on the vehicle behavior [4].

5.2 (1) Proposed Attacks That Induce Collision Between Responses

In this section, we propose attacks that can inject any value of the false responses by generating collisions between the correct and false responses. The attack procedures is shown in **Fig. 3**.

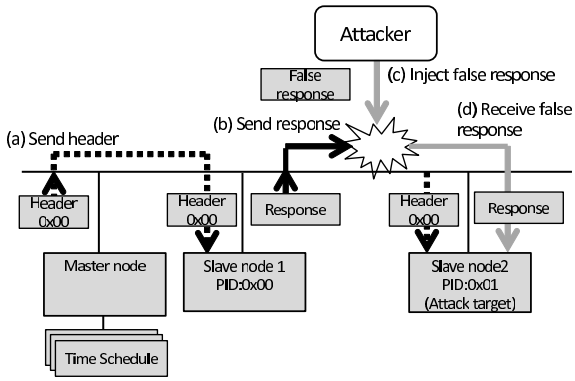

Fig. 3 Attacker induces collision between response sent from slave node 1 and false response.

Here, we consider that slave node 1 sends a response corresponding to the header including the PID (`0x00`) and slave node 2 (attack target) receives it to perform some behavior. The details of the attack method are given below.

(a) The attacker creates a false response that has different values than the correct response from the n-th byte when the attacker wants to forge a response from the $(n+1)$-th byte.

In order to validate the false response, the header, including the PID that indicates slave node 1 sends a response, must be sent in the bus. Then, the attacker waits until the master node sends the header that corresponds to the response of the attack target or the attacker creates the header and sends it.

(b) Corresponding to the header (`0x00` as example), slave node 1 sends a response as shown in Fig. 3.

(c) At this time, the attacker sends a false response at the same time that slave node 1 sends a response. Then, a collision occurs at the n-th byte. Slave node 1 detects the bit error that the bus level is different from the response, and subsequently stops transmitting the response based on the error handling mechanism. Then, the attacker continuously sends the false response from the $(n+1)$-th byte.

We note that when collisions between two responses occur, the value in the bus is electrically 0 in the physical layer, i.e., the dominant level takes priority. Then, the data in the bus where the collision occurs at the i-th byte will be $R_i \& \widetilde{R}_i$ where R_i is the i-th byte of the response sent from the slave node, $\widetilde{R}_i$ is the i-th byte of the false response sent from the attacker, and & is a bitwise `AND`.

(d) Then, slave node 2 receives and stores the false response from the $(n+1)$-th byte as the valid one. We note that the attacker needs to calculate the correct checksum in order that slave node 2 accepts the false response as the valid one. The attacker can calculate the correct checksum by guessing the response transmitted in the bus.

The time chart of the header, the response sent from slave node 1, the false response sent from the attacker and the data in the bus is shown in **Fig. 4**. In the figure, we simply describe the data field in the response. We consider that the response is sent with the least significant byte first. Based on the figure, the attacker can send any value of the false response after the second byte.

As is shown in the above, if the control data is set in the first byte, the attack can not necessarily affect the automotive control because the first byte cannot be changed to the value which the

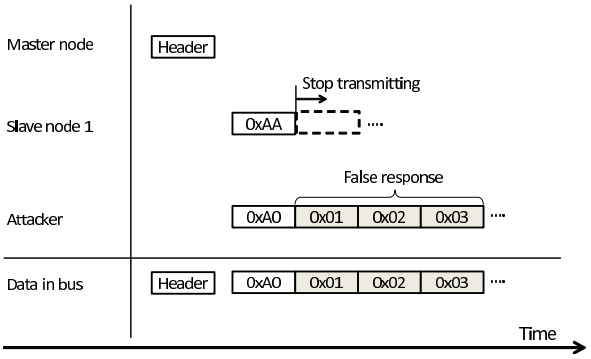

Fig. 4 Time chart of data from master node, slave node 1, attacker, and data in bus when a collision occurs between the first byte of responses. The value of the first byte in bus will be `0xA0` (= `0xAA` & `0xA0`).

attacker likes. However, to the best of our knowledge, it is often the case that the significant control data is set to the byte after the second byte in the data field of the response. Therefore, we think that the proposed attack is practical and it can affect the control system even when the first byte is corrupted by the collision.

In the above attack, the attacker intentionally injects a false response to cause the error handling mechanism to initiate and stops the transmission from the slave node 1. We note that the bit error can be caused by other techniques such as inverting a bit or some bits at the first byte of the response using an electric signal to set the bus level to dominant or recessive[*2]. Then, slave node 1 detects the error and stops the transmission. The subsequent attack procedures are the same as that described above.

5.3 (2) Proposed Attacks That Induce Collision Between Headers

We present a more complex attack method such as an attack that changes the sequence of the response without tampering the time schedule of the master node.

In addition to injecting any value of the message in the bus, we need to cause a collision between headers to change the sequence of the message. The attack procedures are shown in **Fig. 5** and **Fig. 6** and we show the attack to induce any value of the response at any time as follows. In this case, under normal conditions, slave node 2 sends a response that corresponds to the PID (`0x01`); however, we change the sequence of the message where slave node 1 (the PID is `0x00`) sends a response to make slave node 2 receive a false response and perform abnormal behavior.

(a) The attacker sets a false header (including the PID `0x00` as an example) to let slave node 2 (attack target) receive the response[*3]. The master node sends the header (including the PID `0x01`) based on the time schedule.

(b) At this time, the attacker injects a false header (including the PID `0x00`) at the same time that the master node sends the header.

[*2] In Ref. [12], we describe this content in the independent section (Section 4.2.2 in Ref. [12]). However, the essence of the attack is almost the same with the attack using the collisions of the response. Thus, we merge the description in Section 4.2.2 of Ref. [12] in this section.

[*3] In Ref. [12], we describe that the attacker sets a false header which is not assigned to any node after the collisions. However, this is not correct and the attack is not effective because any node does not receive the response if the attacker sets so. Thus, we change the setting of the false header and the attack procedures in the following steps.

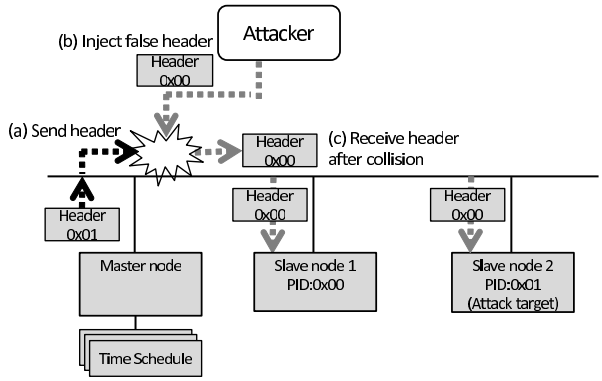

Fig. 5 Attacker injects a false header to send a false message at any timing.

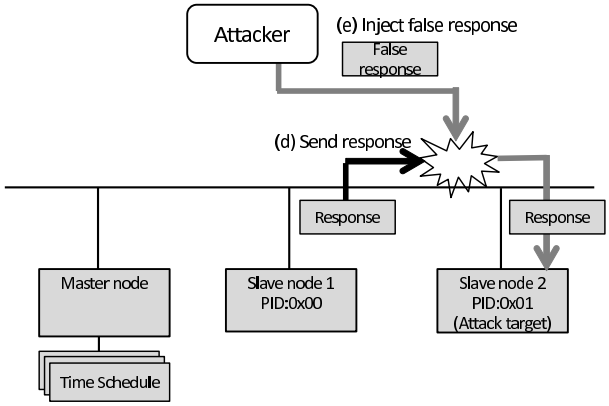

Fig. 6 After collision between headers, attacker injects a false response.

(c) Then, the header in the bus becomes the value `0x00` after the collision and each slave node receives this changed header value.

(d) When the value of the header changes, slave node 1 sends a response corresponding to the received header.

Slave node 2 sends a response under normal conditions; however, the header is changed due to the collision caused by the attacker. Thus, slave node 1, which is not the normal sender, sends a response.

(e) At the same time that slave node 1 sends a response, the attacker injects a false response that is different from the *n*-th byte of the response sent from slave node 1.

(f) In the same way as described in Section 5.2, slave node 2 receives a false response as the valid one.

The time chart of the header, the correct response, the false response and the data in the bus is shown in **Fig. 7**. The figure shows that the attacker can send a false response after changing the sender of the response, i.e., the sequence of the message.

In the above attack, we note that the attacker cannot change any value of the header because the dominant value, 0, takes priority in the physical layer when a collision occurs. However, once the attacker can change the value of the header, he can change the sequence of the transmission of the message without tampering with the time schedule of the master node.

In Sections 5.2 and 5.3, we describe the attacks in which any value of the response can be transmitted at any timing. We note that the attacker can also perform a DoS attack using this attack technique. In fact, the attacker can skip any response in the time schedule by intentionally causing an error because the slave node

stops transmitting a response when it detects an error. Then, the attacker can also perform attacks that simply stop any response in the time schedule.

5.4 Discussions Regarding Safety Mechanisms

In this section, we discuss assumptions regarding safety mechanisms on performing the proposed attack and effectiveness of the attack on them. The safety mechanisms are generally implemented in the automotive system to detect the system failure and recover to the safe status.

We assume that the following general mechanisms are implemented in the system regarding the occurrence of errors such as bit errors.

(1) The node (sender) which detects errors shift to the safe status.

(2) The node (sender) which detects errors sends an abnormal signal which means the occurrence of the errors to all nodes including the receiver. Then, they shift to the safe status.

Regarding (1), although the sender can shift the safe status, other nodes including the receiver cannot detect the error. Thus, it is difficult to mitigate the attacks by this mechanism. Regarding (2), depending on the kinds of the abnormal signal and how to shift the safe status after each node receives its signal, it is possible to mitigate the attacks. The details of how to mitigate the attacks are described in Section 7.3.

In addition to the above safety mechanism, we assume that the safety mechanisms which are commonly used are implemented in the system.

(a) When the slave node (receiver) receives the response including the invalid values which are not defined by the specifications, it discards the response and waits the next header.

(b) When the slave node (receiver) receives the response which means the rapid change when the system does not assume such a change, it discards the response and waits the next header.

(c) The master node continuously sends the same header twice. Thus, the slave node (receiver) verifies whether two responses, corresponding to the header, are the same. If two responses do not match, the receiver discards them and it waits the next header.

The proposed attack does not induce the system failure and the malicious behavior can be caused with the acceptable range of the system. In fact, regarding (a) to (c) in the above, we use the normal values of the response, we do not cause the rapid change of the response and we can send the same response corresponding to the same header twice in the attack. Thus, it is difficult to mitigate the attacks by the mechanisms from (a) to (c).

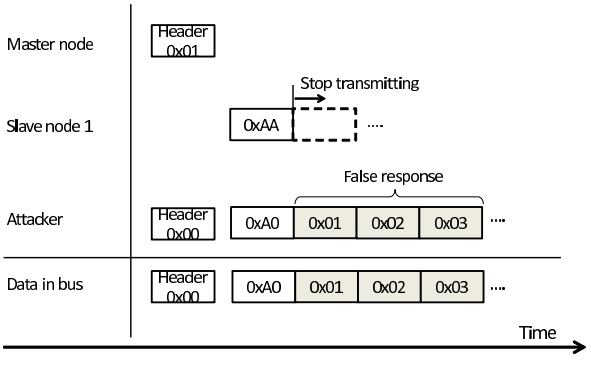

Fig. 7 Time chart of data from master node, attacker, and data in bus when a collision between headers occurs.

5.5 Practical Threats Posed by Proposed Attacks

In this section, we mention the practical threats posed by the proposed attacks. When the significant control data is set after the second byte of the response, the attacker continuously injects the false response using the proposed method through any access to the LIN bus to induce the malicious behavior in the real situations. As concrete examples, we consider the controls of the sliding door and the steering wheel lock in which LIN is generally used. In these cases, we think that the attacker can hold the sliding door open and can affect the control of driving safely by the lock of steering wheel while the vehicle is moving using the proposed attacks. Thus, the malicious behavior caused by the attack results in the significant practical threats.

6. Experimental Results

This section describes the experimental results of the attacks. To verify the feasibility of the proposed attacks, we experimentally analyze the attack when the attacker injects a value of the response as in Section 5.2 as an example using a vehicle microcontroller.

An overview of the experimental configuration is shown in **Fig. 8**. Details of the equipment used in the experiments are given in **Table 1**. The experimental conditions are as follows.

- For both slave nodes and the attacker node, we use the same evaluation boards with the microcontroller and transceiver IC to send a false response at the same time as the slave node.
- We implement the error handling mechanism in the slave node in which it monitors the bus level and stops the transmission when an error is detected.

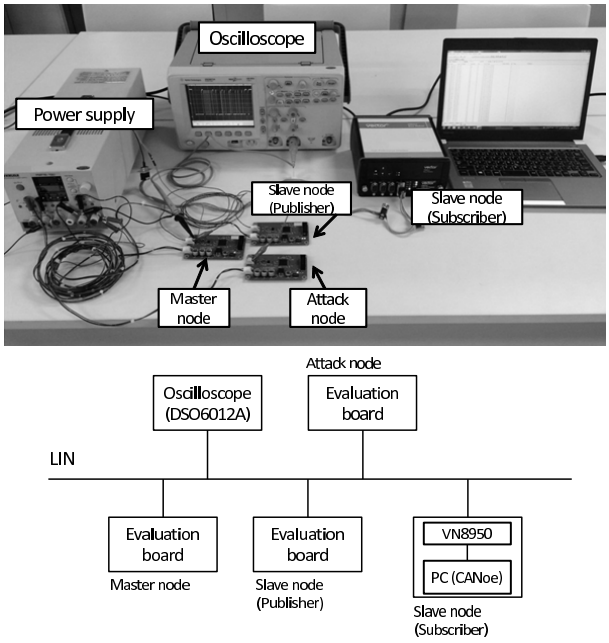

Fig. 8 Overview of experimental configuration.

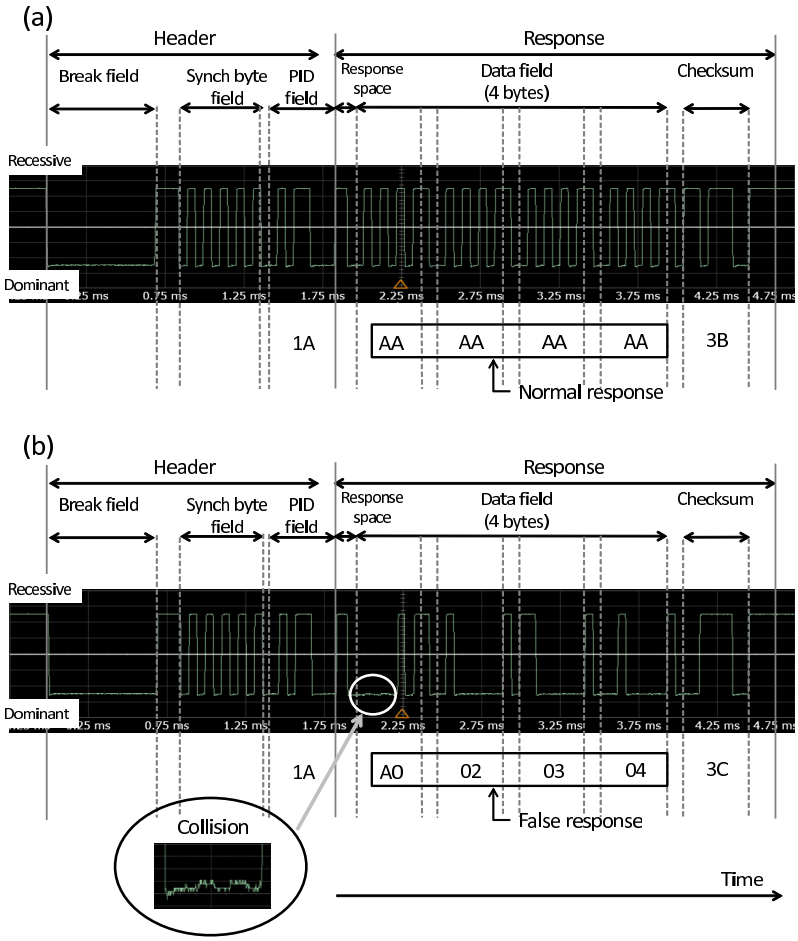

Fig. 9 Experimental results of waveform of header and response transmitted in the bus. (a) Waveform of message frame in normal communications. (b) Waveform of message frame in communication when attack is performed. Collision occurs at the first byte and a false response is transmitted to the bus after the collision.

Table 1 Equipment used in experiment.

Equipment	Product Name and Model Number
Oscilloscope	Agilent DSO6012A
DC Power Supply	Kikusui Electronics Corp., PMP16-1QU
PC (Slave Node)	CPU: Intel Core i7 2.60 GHz, OS: Windows 7 (64 bits), Software: CANoe V8.2

- We use the simulation software CANoe V8.2 from Vector Informatik [19] as the slave node (sender) and use VN8950 which is the hardware interface of the LIN bus including the LIN transceiver [20].
- We use an oscilloscope to measure the waveform of the header, the response, and the false response transmitted in the bus.
- We set the time base to 50 ms and baud rate to 19,200 bps as the LIN conditions, and use LIN protocol version 2.X.
- We set the PID of the header corresponding to the slave node to `0x1A`, the response from the slave node as the 4-byte data `0xAAAAAAAA`, and the false response from the attacker as the 4-byte data `0xA0020304`, as an example. Each response is transmitted with the least significant byte first.
- We use a time schedule that includes the header (the PID `0x1A`) from the master node. The response corresponding to the header is constantly transmitted to the bus. We consider that the attack node sends a false response when it receives the header including `0x1A`.

In the experiment, when the header is sent from the master node, both the slave node and the attack node send responses at the same time, and then, a collision occurs. The experimental results measured using the oscilloscope are shown in **Fig. 9** in the case of normal communications and the commnuications when the attack is performed. In Fig. 9 (a), we show that the response sent from the slave node, `0xAAAAAAAA`, is transmitted normally. On the other hand, in Fig. 9 (b), we observe that false response `0xA0020304` sent from the attacker is transmitted when the first byte collision occurs. We note that we observe a collision, in which two small peaks appear, at the second bit and at the fourth bit of the first byte as shown in Fig. 9 (b). The detailed value is shown in **Fig. 10** when the collisions occur. In the figure, the value in the bus is `0xA0` (= `0xAA` & `0xA0`) after the collision.

To verify that the value of the false response is received by the slave node (subscriber), we observe data measured in the PC. The response in the bus measured using CANoe is shown in **Fig. 11** and we observe that the false response is received at the PC as the valid one.

Therefore, based on the experimental results, we show that the proposed attack is effective using the error handling mechanism.

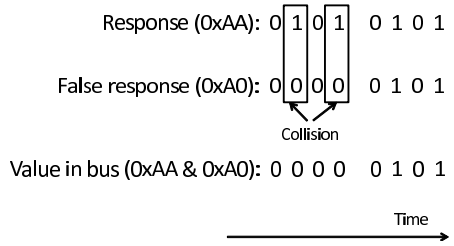

Fig. 10 Collision between the first byte of response `0xAA` (= 1010 1010) and false response `0xA0` (= 1010 0000). Value `0xA0` (= `0xAA` & `0xA0`) is transmitted in the bus. The byte of the response is transmitted with the least significant bit first.

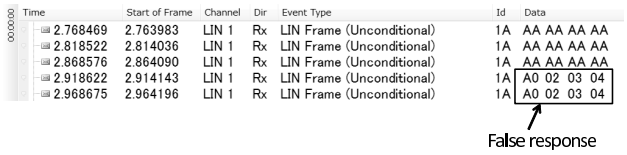

Fig. 11 Response data measured in CANoe.

7. Countermeasures

In this section, we describe countermeasures against the proposed attacks in Section 5[*4].

7.1 Byte Assignment in Response

We set the significant data to the first byte of the response so that values are not changed to those the attacker likes. In the attack, it is difficult to change the first byte of a response that is involved in a collision occurs and it is only possible to change the bytes after a collision. As an example, it is better that we assign the significant byte, which indicates the control system, to the first byte of the response.

7.2 Implementation of Message Authentication Code (MAC)

In the same way as in CAN, a message authentication code (MAC) can protect LIN from an attack and make it difficult for the slave node to receive a false response as the valid one.

However, it is not easy to implement the MAC in LIN from the aspect of the MAC calculation time in the slave node.

7.3 Sending Abnormal Signal When Error is Detected

We propose countermeasures that can be implemented easily based on the normal processing of the LIN protocol.

When the slave node detects that the bus level does not match the sent response while monitoring the bus, it sends an abnormal signal that indicates that an abnormal situation occurred and all nodes shift to a safe status. Moreover, when an error is not continuously detected in the slave node, each node returns from safe status to normal status.

[*4] In Section 6 (3) of Ref. [12], we describe the countermeasures regarding resending the header against the attack using the header collisions (Section 4.2.3 of Ref. [12]). However, the attack using the header collisions in Ref. [12] is not effective as mentioned before. Thus, in this paper, we do not describe the countermeasures of resending the header because it is not appropriate. Furthermore, compared to Ref. [12], we devise a countermeasure which is effective on both attacks in Section 5.2 and Section 5.3 and describe it in Section 7.3 in this paper.

7.3.1 Effectiveness of Countermeasures When Collisions Occur Between Responses

We describe attacks that induce a collision between responses as described in Section 5.2 when the above countermeasure is implemented in LIN to show the effectiveness of the countermeasures.

1. As described in Section 5.2, the attacker injects a false response at the same time that slave node 1 sends a response that corresponds to the header (PID is `0x04` as example) from the master node. At this time, slave node 1 detects the error at the first byte and immediately sends an abnormal signal in which the remaining 8 bytes (7-byte data and 1-byte checksum) are all `0x00`. Even if the attacker sends a false response after the second byte, it is overwritten with the abnormal signal because the dominant 0 takes priority in the bus. Then, the attacker cannot send any value of the false response after the second byte.

2. Next, we describe how to notify all nodes that an abnormal situation occurred and each node returns to normal status when errors do not continuously occur. Based on the time schedule, the master node sends the next header. At this time, slave node 1 sends a response (all 8-byte data are `0x00` and the 1-byte checksum) that indicates that an abnormal situation occurred to the master node regardless of the value of the header.

3. When the master node receives the response described in 2, it sends the specific PID of the header such as `0x00`. When each slave node receives the header including PID `0x00`, they shift to the safe status. Here, the safe status indicates that a part of the functions of the vehicle cannot be active, i.e., the car seat position cannot be adjusted while the car is moving, for example.

4. The master node resends the header (including PID `0x04`) and monitors performances of slave node 1. If slave node 1 sends an abnormal signal to the bus very frequently, the bus for slave node 1 is logically turned off and the vehicle alarm display informs the driver that an abnormal situation occurred.

If slave node 1 normally sends a response that corresponds to the header (including the PID `0x04`) from the master node, the master node decides that slave node 1 is normal and it sends the header including the PID `0xFB` (calculated by 6-bit ID `0x3B` and 2-bit parity `0x11`) as an example, which indicates deactivation of the safe status. When each node receives a header including PID `0xFB`, it shifts the status from safe to normal.

We note that, in step 3, the master node can notify the slave node to shift from safe status using the header which includes a bit indicating the shift to the safe status. As examples, we assign the least significant bit to indicate the shift to the safe status, i.e., bit 1 means normal status and bit 0 means the shift to safe status. After the master node receives an abnormal signal from slave node 1, the master node sends the next header including PID `0x10` as an example based on the time schedule. Each slave node detects that the least significant bit of the PID is 0. It then shifts the status from safe to normal. When slave node 1 normally sends a response and the master node decides that slave node 1 should return to normal status, the master node sends the next header including the PID in which the least significant bit is 0 based on the time schedule. In this case, each slave node can return to normal status.

7.3.2 Effectiveness of Countermeasures When Header Collision Occurs

Here, we describe attacks that induce a collision between headers as described in Section 5.3 when the above countermeasure is implemented in LIN to show the effectiveness of the countermeasures.

As described in Section 5.3, the attacker injects a false header at the same time the master node sends a header (including the PID `0x05` as an example). At this time, the master node detects an error in which the value of the header in the bus is different from that it sent while monitoring the bus. Then, the master node sends a response in which the 8-byte data are all `0x00`. Therefore, if the attacker sends a false response after injecting a false header, the former is overwritten with the responses in which all 8-byte data are `0x00` and the attacker cannot send any value of the false response. The subsequent processing is the same as that 3 and 4 in Section 7.3.1.

Thus, the above countermeasures protect from the attack described in Sections 5.2 and 5.3 so that an abnormal signal overwrites the false response from the attacker. Furthermore, this countermeasure includes a function in which each node can return to normal status from the safe status when an abnormal situation does not continuously persist.

8. Conclusions

This paper presented a security analysis of LIN. We presented case studies of attacks that can induce malicious behavior by injecting any value of the false response in time using the error handling mechanism. We performed experimental analysis of the proposed attacks and verified the feasibility of the attacks using a vehicle microcontroller. The results showed that the attacks were successful, i.e., we sent a false response in the LIN bus and the receiver node accepted the false response as the valid one. Furthermore, we presented countermeasures that can be easily implemented based on the processing of the LIN protocol. We pointed out a vulnerability if the error handling mechanism, which is not generally determined in the protocol specification, was simply designed. Depending on the situation, we consider that the LIN protocol specification requires updating in regard to security. We consider that the attack concept and the countermeasures presented in this study can be applied to other in-vehicle protocols. We believe that this study will contribute to improvements in security of vehicle communications.

References

[1] Koscher, K., Czeskis A., Roesner, F., Patel, S., Kohno, T., Checkoway, S., McCoy, D., Kantor, B., Anderson, D., Shacham, H. and Savage, S.: Experimental Security Analysis of a Modern Automobile, 2010 IEEE Symposium on Security and Privacy (SP), *IEEE Computer Society*, pp.447–462 (2010).

[2] Hoppe, T., Kiltz, S. and Dittmann, J.: Security Threats to Automotive CAN Networks – Practical Examples and Selected Short-Term Countermeasures, *International Conference on Computer Safety, Reliability, and Security* (*SAFECOMP 2008*), *LNCS 5219*, pp.235–248 (2008).

[3] Wolf, M., Weimerskirch, A. and Paar, C.: Secure In-Vehicle Communication, *Embedded Security in Cars*, Lemke, K., Paar, C. and Wolf, M. (Eds.), pp.95–109, Springer-Verlag Berlin Heidelberg (2006), ISBN-10 3-540-28384-6 (Print).

[4] Valasek, C. and Miller, C.: Adventures in Automotive Networks and Control Units, Technical White Paper, IOActive (2014).

[5] Hoppe, T., Kiltz, S. and Dittmann, J.: Automotive IT-Security as a Challenge: Basic Attacks from the Black Box Perspective on the Example of Privacy Threats, *International Conference on Computer Safety, Reliability, and Security* (*SAFECOMP 2009*), *LNCS 5775*, pp.145–158 (2009).

[6] Bittl, S.: Attack Potential and Efficient Security Enhancement of Automotive Bus Networks Using Short MACs with Rapid Key Change, *Nets4Cars/Nets4Trains/Nets4Aircraft 2014, LNCS 8435*, pp.113–125 (2014).

[7] Herrewege, A.V., Singelee, D., Verbauwhede, I.: CANAuth - A Simple, Backward Compatible Broadcast Authentication Protocol for CAN bus, *Proc. Workshop on Embedded Security in Cars* (*escar 2011*) (2011).

[8] Wolf, M.: Security Engineering for Vehicular IT Systems, *Vehicular Security Mechanism*, Vieweg+Teubner (2009), ISBN: 978-3-8348-0795-3 (Print).

[9] Larson, U.E., Nilsson, D.K. and Jonsson, E.: An Approach to Specification-based Attack Detection for In-Vehicle Networks, *2008 IEEE Intelligent Vehicles Symposium*, pp.220–225, IEEE Computer Society (2008).

[10] AUTOSAR: Specification of Module Secure Onboard Communication AUTOSAR Release 4.2.2 (online), available from ⟨http://www.autosar.org/fileadmin/files/releases/4-2/software-architecture/safety-and-security/standard/AUTOSAR_SWS_SecureOnboardCommunication.pdf⟩ (accessed 2016-05-12).

[11] Stelzer, J.: LIN bus emerging standard for body control apps (online), available from ⟨http://www.eetasia.com/ARTICLES/2004SEP/B/2004SEP16_NTEK_ID_TA.pdf?SOURCES=DOWNLOAD⟩ (accessed 2016-05-12).

[12] Takahashi, J., Aragane, Y., Miyazawa, T., Fuji, H., Yamashita, H., Hayakawa, K., Ukai, S. and Hayakawa, H.: Automotive Attacks and Countermeasures on LIN-Bus, *Symposium on Cryptography and Information Security*, Kumamoto, Japan (2016), 4F2-5, 8 pages (in Japanese).

[13] FUJITSU AN704-00007-2v0-E, 16-BIT MICROCONTROLLER F^2MC-16FX Family MB96600 series, How to control LIN communication.

[14] National Instruments, Introduction to the Local Interconnect Network (LIN) Bus, White Paper, Publish Date: Nov. 03 (2011).

[15] Rippel, E.: Embedded Security Challenges in automotive designs, *Proc. Workshop on Embedded Security in Cars* (escar 2008) (2008).

[16] ISO/DIS 17987-3, Road vehicles – Local Interconnect Network (LIN) – Part 3: Protocol specification (Nov. 2013).

[17] Nilsson, D.K., Larson, U.E., Picasso, F. and Jonsson, E.: A First Simulation of Attacks in the Automotive Network Communications Protocol FlexRay, *Proc. International Workshop on Computational Intelligence in Security for Information Systems* (*CISIS 2008*), *Advances in Soft Computing*, Vol.53, pp.84–91 (2009).

[18] STMicroelectronics, Microcontroller Division Applications, AN1278 APPLICATION NOTE LIN (LOCAL INTERCONNECT NETWORK) SOLUTION.

[19] Vector Informatik, CANoe/DENoe V8.2.

[20] Vector Informatik, VN8950: CAN/LIN/J1708 module with analog/digital IO expandability.

Junko Takahashi received her B.S. and M.S. degrees in physics from Waseda University, Japan, in 2004 and 2006, respectively, and Ph.D. degree in engineering from the University of Electro-Communications, Japan, in 2012. She joined NTT Information Sharing Platform Laboratories, Nippon Telegraph and Telephone Corporation in 2006. Currently, she is a researcher with NTT Secure Platform Laboratories. She is a member of IEICE and IPSJ. Her main research interest is the security of embedded systems such as side-channel analysis and automotive security. She was awarded the SCIS 2008 Paper Prize.

Yosuke Aragane is Senior research engineer, supervisor, NTT R&D planning department. He received his M.S. and Ph.D. degrees from Tokyo Institute of Technology in 1997 and 2005. He joined NTT Multimedia Network Laboratories in 1997, where he worked on intelligent transportation systems. Since 2003, he has been with NTT Information Sharing Platform Laboratories and NTT Secure Platform Laboratories focusing on cybersecurity research. During 2008–2011, he was also with the IT Innovation department, NTT East Corporation. He was a director of NTT-CERT, the representative CSIRT of NTT group. He is a member of IEEE, ACM, IPSJ, and IEICE.

Toshiyuki Miyazawa is Manager, Security Strategy Section, NTT Technology Planning Department. He received his B.E. and M.S. degrees in mathematics from Waseda University, Tokyo, in 2000 and 2003, respectively. Since joining NTT Information Sharing Platform Laboratory in 2003, he has been engaged in R&D of information security, especially of public key cryptography and security protocols. From 2008 to 2011, he was with the IT Innovation Department at NTT EAST. He is a member of the Japan Society for Industrial and Applied Mathematics. He received the SCIS Paper Award from the Institute of Electronics, Information and Communication Engineers (IEICE) in 2007.

Hitoshi Fuji is Executive Manager, Data Security Project, NTT Secure Platform Laboratories. He received his B.E. and M.E. degrees from Tokyo University of science in 1991 and 1993, respectively. Since joining NTT in 1993, he had been engaged in research on software engineering, network security and information security. He also earned Ph.D. in informatics.

Hirofumi Yamashita received his B.S. and M.S. degrees in Mechanical Engineering from Mie University, Japan, in 2000 and 2002, respectively. He joined DENSO Corporation in 2002. Currently, he is an engineer with DENSO Corporation. His main research interest is the Automotive Cyber Security form 2012.

Keita Hayakawa received his B.S. and M.S. degrees in Electronic Engineering from Nagoya University, Japan, in 2009 and 2011, respectively. He joined DENSO Corporation in 2011. Currently, he is an engineer with DENSO Corporation. His main research interest is the Automotive Cyber Security form 2014.

Shintaro Ukai received his B.S. and M.S. degrees in Electronic engineering from Ritsumeikan University, Japan, in 2012 and 2014, respectively. He joined DENSO Corporation in 2014. Currently, he is an engineer with DENSO Corporation. His main research interest is the Automotive Cyber Security form 2014.

Hiroshi Hayakawa received his B.S. and M.S. degrees in Electronic engineering from Osaka University, Japan, in 1991 and 1993, respectively. He joined DENSO Corporation in 1993. Currently, he is an engineer with DENSO Corporation. His main research interest is the Automotive Cyber Security form 2000.