*electronics*

*Article*

# Proposal for the Sixth Error Type for Cyberattack Detection and Defense in CAN Protocol

**Yunkeun Song [1], Yongeun Kim [2], Yousik Lee [3,*] and Samuel Woo [4,*]**

[1] Department of Computer Science, Dankook University, Yongin 16890, Republic of Korea; yunkeun.song@dankook.ac.kr
[2] Synergytion Co., Ltd., Cheonan 31156, Republic of Korea; synergytion@gmail.com
[3] Department of Information Security, Soonchunhyang University, Asan 31538, Republic of Korea
[4] Department of Software Science, Dankook University, Yongin 16890, Republic of Korea
* Correspondence: yousik.lee@sch.ac.kr (Y.L.); samuelwoo@dankook.ac.kr (S.W.)

**Abstract**

Having long served as the backbone of automotive communication, the Controller Area Network utilizes error handling mechanisms under the ISO 11898 standard for communication reliability. However, these legacy error types do not explicitly distinguish between simple electrical noise and malicious intent. To address this structural limitation, we propose a sixth error type as a specialized protocol extension considering cybersecurity along with an error frame designed to notify other controllers and the driver of cybersecurity attacks. By defining a specific detection logic capable of identifying impersonation and replay attacks and introducing a specialized frame structure, this study enables the data link layer to take immediate defensive action without complex cryptographic overhead. Through FPGA based prototyping and Vector CANoe testing, we demonstrated that this mechanism successfully invalidates malicious attempts while preserving compatibility with the existing CAN error-handling mechanism. This research argues that cybersecurity can no longer be treated as an add-on but should be embedded within the protocol itself. Our findings provide a technical foundation for the next evolution of the ISO 11898 standard and toward security integrated CAN communication.

**Keywords:** Controller Area Network (CAN); error frame; cybersecurity; in-vehicle network (IVN); real-time invalidation



Academic Editors: Kailong Zhu, Xuehu Yan, Jiazhen Zhao and Xin Zhou

Received: 11 March 2026
Revised: 6 April 2026
Accepted: 14 April 2026
Published: 17 April 2026



## 1. Introduction

Modern vehicles require reliable inter-system communication to support advanced driver assistance systems (ADAS), connectivity, and safety functions [1]. Although high-performance protocols such as Automotive Ethernet have been introduced to accommodate increasing bandwidth demands, the Controller Area Network (CAN) remains the most widely adopted in-vehicle communication protocol due to its cost efficiency and proven robustness [2,3]. CAN employs a non-destructive bitwise arbitration scheme in its multi-master communication architecture, ensuring that when multiple nodes attempt to transmit simultaneously, the highest priority message is delivered without collisions. In addition, CAN specifies error detection mechanisms and fault confinement rules based on error counters and node states, thereby ensuring the reliability and fault tolerance of communication. These mechanisms prevent faults originating from a single node from spreading across the network, ensuring stable communication even in demanding automotive environments [4].

Since its initial standardization as ISO 11898 in 1993, CAN has undergone revisions in 2003, 2015, and 2024 to address the evolving demands of in-vehicle communication [5–7]. During these updates, the CAN with flexible data rate (CAN FD) frame and the CAN with Extended Data Length (CAN XL) frame were introduced to expand the data field size and improve transmission efficiency. However, these changes have been limited to enhancements in payload capacity and data rate. Fundamentally, the reliability-oriented design philosophy has remained unchanged, with error detection and fault confinement rules preserved. As a result, the protocol provides no specification for detecting cybersecurity threats or delivering this information to other systems or to the driver.

Building on these limitations, numerous studies over the past decade have explored attack techniques that exploit the structural weaknesses of the CAN error handling mechanisms defined in ISO 11898, as well as detection and mitigation strategies. Early research demonstrated that spoofing-based bus-off attacks could force an ECU into the bus-off state, thereby disconnecting it from the network [8]. To counter this, intrusion detection systems were proposed that continuously monitor the CAN bus to detect abnormal error frame patterns or the transmission of identifiers not originating from legitimate senders [9]. However, these defenses were inherently reactive, identifying attacks only after the victim Electronic Control Unit (ECU) had already transitioned into the bus-off state. Later, more evasive attack techniques were introduced, such as error-frame skipping-attacks [10] and precise synchronization with the victim's transmission [11], while defensive strategies proposed adding pre-computed transmission delays between frames [12] or introducing additional cybersecurity error states [13]. Despite these advances, existing defense mechanisms remain constrained by detection latency, deployment complexity, and hardware dependency.

In contrast to international standardization and academic research, industry-led initiatives have focused on defining security specifications for the CAN protocol. Under the leadership of the Security Interest Group (SIG) within the CAN in Automation (CiA) consortium, the Controller Area Network Security (CANSec) framework has been proposed to introduce message authentication, encryption, and replay protection for CAN XL [14–16]. The focus on CAN XL arises from its extended data field and higher bandwidth, which offer the sufficient payload capacity required for cryptographic operations that were previously constrained by the limited message size of classic CAN or CAN FD. Although CANSec represents a significant step toward embedding security primitives directly into the protocol, its current specification is exclusively defined for CAN XL, leaving legacy protocols—which still dominate existing vehicle architectures—vulnerable.

While academia and industry continue to propose technical solutions to overcome the cybersecurity limitations of CAN, both regulatory frameworks and international standards on functional safety and cybersecurity are evolving in parallel. Annex 5 of UN regulation no.155 provides a comprehensive definition of cybersecurity threats to vehicles and mitigation measures, but its focus remains on administrative and technical responses from the perspective of vehicle manufacturers rather than on direct driver awareness [17]. ISO 26262 [18] aims to reduce risks arising from faults or failures that affect vehicle safety to an acceptable level. As part of its safety mechanisms, the standard requires that detected malfunctions be communicated to the driver through warnings or status indications, allowing prompt corrective actions. Furthermore, the Safety of the Intended Functionality (SOTIF) standard, ISO 21448 [19], addresses risks related to the intended functionality of ADAS and perception technologies, even in the absence of hardware or software failures. It goes beyond merely providing warnings, instead requiring that the driving task be handed over to the driver via the Human-Machine Interface (HMI), thus underscoring the HMI as one of its core concepts. Together, ISO 26262 and ISO 21448 clearly demonstrate the need for HMIs to ensure driver intervention in unintended situations. In contrast, the

only dedicated automotive cybersecurity standard, ISO/SAE 21434, defines engineering requirements and provides a well-structured development process, but does not explicitly account for HMIs in the event of cybersecurity incidents [20].

In today's increasingly complex automotive environment, cyberattacks can trigger unintended malfunctions of vehicle functions, making it essential not only to detect, record, and mitigate such incidents at the vehicle manufacturer level but also to communicate them transparently to drivers in real time to prevent or mitigate accidents. In particular, mechanisms are required that can distinguish between general communication reliability anomalies and cybersecurity anomalies and notify drivers in a clear and timely manner when a cybersecurity threat is identified. However, current regulations, international standards, and industrial technical proposals do not provide such differentiation and driver notification capabilities.

In this paper, we propose a cybersecurity-aware sixth error type and error frame Structure, a novel protocol-level mechanism that extends the five error types defined in ISO 11898, enabling the distinction between cybersecurity-oriented communication anomalies and reliability-oriented communication errors, thereby providing a basis for real-time driver notification of cybersecurity attacks. This paper is structured as follows. Section 2 provides essential background on the CAN protocol, including bus topology, message/frame structure, and error handling mechanisms. Section 3 analyzes representative cybersecurity attacks and prior studies on detection and mitigation that motivate our study and identifies their limitations. Section 4 proposes a cybersecurity-aware sixth error type and error frame Structure designed for cybersecurity attack detection and notification. Section 5 describes the activities conducted to validate the feasibility of the proposed mechanism. Section 6 discusses the limitations of the proposed mechanism. Finally, Section 7 summarizes the contributions and concludes the paper.

## 2. Background

CAN is an in-vehicle network protocol developed in the late 1990s by Bosch Gmbh [21]. At the time of its development, in-vehicle networks were considered isolated systems, disconnected from external entities; thus, the protocol did not incorporate any security features. However, modern vehicles actively communicate with external vehicle entities, making them increasingly vulnerable to cyberattacks. In this section, we explain the characteristics and message structure, the error handling mechanisms, and cyber attacks on the CAN protocol in order to provide a comprehensive understanding of the proposed mechanism.

### 2.1. CAN Bus Topology

CAN has been developed to replace the complex wiring harness with two twisted cables called CAN High (CAN_H) and CAN Low (CAN_L). The logical state in CAN is determined by the voltage difference between CAN_H and CAN_L. When the voltage difference is 0 V, the bus is in the recessive state, representing a logical '1'. Conversely, when the voltage difference is 2V, the bus is in the dominant state, representing a logical '0'. The Electronic Control Unit (ECU) sequentially generates dominant and recessive states to construct a data frame containing the desired data. Additionally, the network topology of CAN follows a bus topology, and communication occurs via a multicast mechanism, where a message transmitted by one node is received by all nodes on the network [4].

### 2.2. Message Structures of CAN

The CAN protocol defines the data frame format used for communication between nodes in a CAN network. As shown in Figure 1, a CAN data frame consists of the Start of Frame (SOF), Arbitration, Control, Data, Cyclic Redundancy Check (CRC), Acknowl-

edgment (ACK), and End of Frame (EOF). Following the multimaster concept of CAN, since any node can transmit a message when the bus is in an idle state, collisions may occur. To avoid collisions, CAN adopts CSMA/CD+AMP (Carrier Sense Multiple Access/Collision Detection with Arbitration on Message Priority), and the arbitration field determines the transmission priority among nodes. This field contains the unique identifier of each function, which allows the receiving ECU to identify the transmitting ECU.

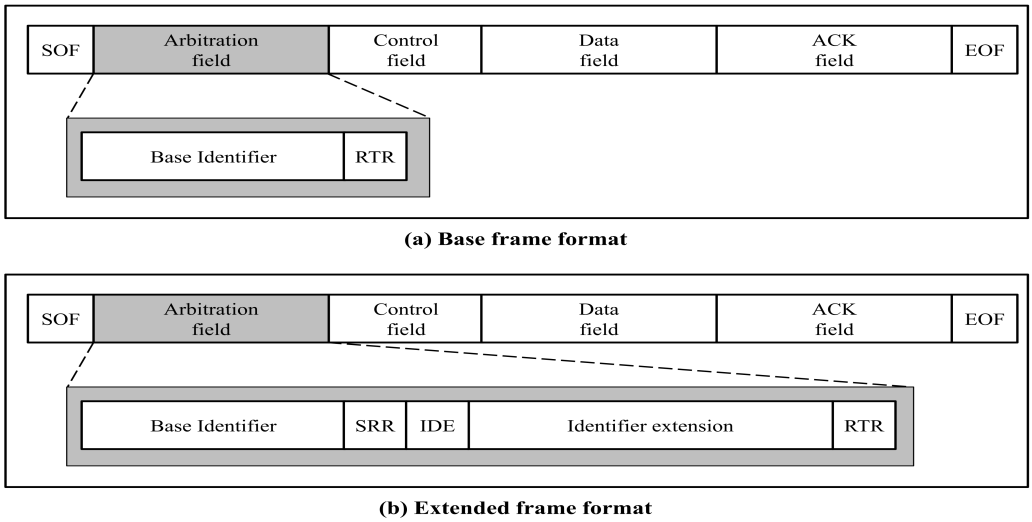



**Figure 1.** CAN 2.0A base frame format.

### *2.3. Error Handling of CAN*

The CAN implements a robust error detection and handling mechanism to ensure data consistency across all nodes. In CAN communication, a received frame is considered valid only if no node detects an error throughout the entire duration of the frame. When an error occurs during communication, a node in the network may generate an error frame. As illustrated in Figure 2, an error frame comprises an error flag, an echo error flag, and an error delimiter field. The node that detects the error transmits six consecutive dominant or recessive bits as the error flag, depending on its status. This is followed by an echo error flag of zero to six bits, which accounts for the overlap of flags from different nodes. Each node recognizes the end of the error frame through eight consecutive recessive bits of the error delimiter. Upon receiving an error frame, the transmitting node ceases data transmission and re-enters the arbitration phase after inter-frame space.

| **Error flag** (6 bits) | **Echo error flag** (0 to 6 bits) | **Error delimiter** (8 bits) |
|---|---|---|

**Figure 2.** Structure of the error frame in ISO 11898.

CAN utilizes five error types as described in Table 1: bit error, stuff error, CRC error, form error, and ACK error. Each node is capable of detecting different types of errors based on their respective error characteristics. Since the CRC error is determined after all data fields have been received, the error frame is transmitted following the ACK delimiter. In contrast, for errors other than CRC errors, the error frame is transmitted immediately after the bit in which the error occurs. Table 1 shows the error detection and handling of CAN.

**Table 1.** Description of the CAN error type.

| Error Type | Description | Detected by | Error Frame Starting Bit |
|---|---|---|---|
| Bit error | An error occurs when the transmit and receive bit are different, except during arbitration. | Transmitting ECU | Next bit of occurrence bit |
| Stuff error | An error occurs when six or more consecutive bits are received. | Receiving ECU | Next bit of occurrence bit |
| CRC error | An error occurs when the CRC value calculated by the receiving node differs from the received CRC field value. | Receiving ECU | Next bit of ACK delimiter |
| Form error | An error occurs when an invalid bit is in the fixed-form bit field. | Receiving ECU | Next bit of occurrence bit |
| ACK error | An error occurs when the value of the ACK field is a recessive state ('1'). | Transmitting ECU | Next bit of occurrence bit |

### *2.4. Cyberattacks on CAN*

To ensure network flexibility, CAN does not require any software or hardware modifications to existing nodes when a new node is added [4]. Additionally, all data frames are broadcast, and there is no built-in authentication mechanism. Consequently, an attacker can easily join the network and receive messages or transmit manipulated ones. For this reason, one of the most easily executed attacks is replay attacks and impersonation attacks.

#### 2.4.1. Replay Attack

A replay attack is a cyberattack in which an attacker retransmits a previously captured CAN message that was originally sent by a legitimate ECU, thereby inducing one or more receiving ECUs to accept the message as legitimate and act on it [22]. In practice, an attacker can access the in-vehicle network through the OBD-II port or similar interfaces and monitor CAN traffic using analysis tools. The attacker can then record messages whose payload values change in response to driver actions (e.g., braking or engine-off events) and later replay the recorded messages to trigger similar effects without performing the corresponding physical actions [23]. Such replay attacks can be mitigated by ensuring message authenticity and freshness; However, in CAN, applying these protections is challenging because the limited payload size of CAN data frames leaves little room to carry such security-related information.

#### 2.4.2. Impersonation Attack

An impersonation attack is a cyberattack in which an attacker forges CAN messages by spoofing a legitimate message identifier (CAN ID) that is normally transmitted by a trusted ECU, causing other ECUs to accept forged messages as if they originated from that legitimate source. After gaining bus access and observing CAN traffic, the attacker can craft malicious messages with the target CAN ID and transmit them over the network to command desired vehicle actions. Such impersonation attacks can be mitigated by strengthening message/source authenticity (e.g., authentication mechanisms), subject to the same payload-size limitations of CAN data frames discussed above.

## 3. Related Work

Many studies have been conducted on attack techniques that exploit the structural limitations of the CAN error handling mechanisms defined in ISO 11898, along with the corresponding detection and response strategies.

Cho and Shin [8] proposed an attack technique that forces a victim ECU into the bus-off state by spoofing a recessive bit as a dominant bit during data field transmission, thereby inducing repeated bit errors and causing the transmit error counter (TEC) to

accumulate. However, this attack technique exhibits two distinctive patterns: the F1 pattern, characterized by consecutive error frames with the same identifier, and the F2 pattern, in which a message bearing the same identifier is successfully received on the CAN bus even though it was not transmitted by the victim ECU. These patterns make the attack technique easily detectable. Leveraging these characteristics, Longari et al. [9] proposed an intrusion detection system (IDS) that continuously monitors CAN traffic and estimates each ECU's TEC in real time. By detecting F1 and F2 patterns, this method enables the identification of spoofing-based bus-off attacks. However, because the IDS can only detect an attack after the target ECU's TEC has already increased, it cannot prevent the victim ECU from transitioning into the bus-off state.

Bloom [10] modified the bus-off attack technique introduced by Cho and Shin [8] and proposed the WeepingCAN attack. The key idea of WeepingCAN is to reduce the observable features of the original attack that are used as detection cues. Specifically, WeepingCAN combines (i) a skipping strategy, in which selected attack cycles are deliberately omitted to avoid generating consecutive error-frame patterns, and (ii) recessive-bit injection. Through this combination, WeepingCAN suppresses the F1 and F2 patterns that are characteristic of the original bus-off attack, and thus can evade F1/F2-based intrusion detection systems (IDSs), such as the one proposed by Longari et al. [9]. However, despite this improved stealthiness, WeepingCAN achieves a success rate of only about 75%. Subsequently, Agbaje et al. [11] improved the attack by introducing a zero-phase synchronization technique that detects and aligns with the victim ECU's start-of-frame (SOF). This synchronization allows the attacker to more precisely time the attack, increasing the success rate to over 90%. Furthermore, they demonstrated that the attack can be extended in a transitive manner, enabling an adversary to sequentially force multiple ECUs into the bus-off state rather than targeting only a single ECU.

Serag et al. [12] proposed a defense mechanism that leverages the inter-frame space (IFS) between CAN data frames by introducing pre-computed transmission delays for each CAN message identifier, thereby enabling both the detection and prevention of bus-off attacks. However, this method requires installing software agents on all ECUs, deploying a trusted officer node to collect all communication messages, and pre-distributing secret keys, which imposes considerable deployment constraints.

Ohira et al. [13] proposed a mechanism that introduces three additional security error states, conceptually similar but distinct from the standard error states defined in ISO 11898. By default, an ECU operates in the security error-active state. Once an attack is detected, the ECU transitions to the security error-passive state, during which it is prohibited from transmitting regular CAN data but can still send warning messages to inform other nodes. If further malicious activity is detected, the ECU is forced into the security bus-off state, thereby isolating it completely from the CAN bus. This mechanism is implemented at the Linux kernel level, where an allowlist-based detector and a transmission-similarity detector are employed to detect both malicious-ID and benign-ID DoS attempts, ultimately preventing compromised ECUs from transmitting further messages. However, this approach requires that a Linux kernel-level security module be applied to every ECU and remains ineffective against sophisticated attacks that accurately emulate legitimate message identifiers and transmission intervals.

## 4. A Cybersecurity-Aware Sixth Error Type and Error Frame Structure

Unlike conventional application-layer security mechanisms, our proposed approach operates directly at the data-link layer of the ISO 11898 CAN standard. By implementing minimal modifications to the existing protocol, this mechanism enables the real-time detection and response to cybersecurity attacks while facilitating their notification to other

ECUs on the bus. This suggests that cybersecurity-aware functionality can be incorporated into the CAN protocol itself.

Due to the broadcast nature of CAN and its lack of built-in authentication mechanisms, which are fundamental protocol-level limitations, adversaries can effortlessly execute both replay and impersonation attacks. As discussed in Section 2, while the execution methods of these attacks differ, they share a core characteristic: hijacking a valid CAN frame to trigger malicious behaviors at a time intended by the attacker. Both attacks result in the injection of unintended malicious data frames into the bus, despite the absence of an actual transmission from the legitimate ECU.

While a spoofed data frame looks legitimate to all other nodes on the bus, only the ECU designed to transmit the CAN ID can detect the attack during the arbitration phase. For instance, if the Engine ECU is the only node designed to transmit CAN ID 0x3F, the appearance of this ID on the bus—even though the Engine ECU has not actually initiated the transmission of a data frame with that ID—serves as conclusive evidence of an attack. Therefore, upon detecting its own identifier, the victim ECU can instantly classify it as a cyberattack and respond by transmitting an error frame to invalidate the threat before the data frame's payload is broadcast. Based on this detection principle, we propose a novel cybersecurity error type and an error frame configuration designed to indicate a cybersecurity-related error, thereby allowing the victim ECU to promptly invalidate a suspicious frame and propagate the detection result to other ECUs on the bus.

### *4.1. Definition of the Sixth Error Type*

Error handling in the CAN protocol relies on five predefined error types: bit error, stuff error, CRC error, form error, and ACK error. These standard errors are primarily designed to detect communication reliability issues caused by physical layer faults or electrical noise. In this study, we additionally propose a sixth error type specifically designed to detect cyberattacks, such as replay and impersonation attacks, and designate it as the cybersecurity error.

The receiving ECU identifies a cybersecurity error when it detects its own CAN ID along with a dominant value in the Remote Transmission Request (RTR) field on the CAN bus. A dominant value in the RTR field indicates that the message is a Data Frame containing a payload. Therefore, detecting this condition without the ECU having initiated the transmission itself serves as conclusive evidence of an attacker attempting to impersonate the victim or replay captured messages. The receiving ECU then generates and transmits an error frame at the bit immediately following the arbitration field, specifically the CAN ID and the RTR field. This action invalidates the malicious frame before its payload is broadcast while simultaneously notifying other ECUs on the bus that a cybersecurity attack is currently occurring.

Table 2 shows the description, the detecting ECU, and the generation timing of the error frame for the proposed cybersecurity error.

**Table 2.** Description of the proposed sixth CAN error type for cyberattacks.

| Error Type | Description | Detected by | Error Frame Starting Bit |
|---|---|---|---|
| Cybersecurity error | When a message is received that raises suspicion of a replay or impersonation attack—specifically, when the ECU receives a message bearing the same identifier as its own and the RTR (Remote Transmission Request) bit is dominant. | Receiving ECU | Next bit of arbitration field |

### *4.2. Structure of the Error Frame*

Upon detecting the proposed cybersecurity error, the ECU immediately generates and transmits an error frame to invalidate the ongoing attack and notify other ECUs on the bus that cybersecurity attack has been detected. To ensure that the ECUs on the network explicitly identify this event as a cybersecurity attack rather than a communication reliability error, we propose a cybersecurity-aware error frame structure. Specifically, the proposed structure extends the conventional CAN error frame, which consists of error flags and an error delimiter, by introducing an additional cybersecurity flag field. Figure 3 shows the structure of the cybersecurity-aware error frame.

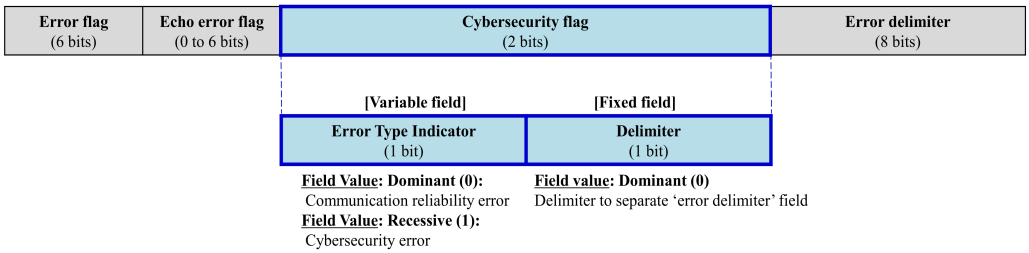



**Figure 3.** Structure of the proposed cybersecurity-aware error frame.

The first bit of the cybersecurity flag, referred to as the Error Type Indicator, is a variable field used to classify the type of detected error. Setting this bit to dominant (0) signifies a communication reliability error, representing standard protocol violations defined in ISO 11898, while a recessive (1) value indicates a cybersecurity error, such as an impersonation or replay attack. This differentiation allows network nodes to explicitly identify whether the triggered error frame originates from a conventional physical fault or a cybersecurity attack.

The second bit of the cybersecurity flag, referred to as the delimiter, is a fixed field consistently set to dominant (0). It serves to distinguish the cybersecurity flag from the subsequent error delimiter field. This bit is particularly vital during a cybersecurity error event when the error type indicator is set to recessive (1). Without fixing this bit to the dominant state, a sequence consisting of the recessive indicator, the 8-bit error delimiter, and the 3-bit IFS would result in 12 consecutive recessive bits on the bus. Since an ECU interprets 11 consecutive recessive bits as "Bus Idle", the absence of this bit would cause the ECU to prematurely misinterpret the bus state as idle before the error handling is complete, leading to synchronization failures. Therefore, by fixing this field to dominant, an explicit boundary is preserved, ensuring that the error delimiter is reliably identified.

Integrating the cybersecurity flag into the error frame rather than the data frame is a strategic choice focusing on immediate blocking upon detecting a cybersecurity attack while minimizing the scope of protocol modification. While altering the data frame structure would require an extensive redesign of payload configurations and CRC calculation logic, the error frame remains a transient sequence triggered only during anomalies; this allows the security mechanism to be implemented independently within the exception-handling path without affecting data frame transmission. Specifically, by immediately transmitting an error frame following the arbitration phase, the system effectively halts the transmission of the remaining data frame intended for the attack and can explicitly notify other nodes that a cybersecurity attack has occurred.

### *4.3. Vehicle-Level Response Strategy for Cybersecurity Errors*

Unlike the five conventional types of CAN errors defined in ISO 11898, cybersecurity errors, specifically those arising from impersonation or replay attacks, necessitate

a distinct response strategy to ensure vehicle safety. Our study proposes the following countermeasures for error state management:

- Immediate Transition to Bus-off and Limp-home Mode: Upon detecting a cybersecurity error, the victim ECU should immediately transition to the bus-off state, bypassing the incremental error counting process. This rapid isolation is critical because sustained contention between the legitimate ECU and an attacker can lead to a Denial-of-Service (DoS) condition, potentially saturating the bus bandwidth and compromising critical vehicle functions. Upon entering bus-off, the vehicle should engage in a limp-home mode to maintain minimal necessary operation. Additionally, a dedicated Cybersecurity Error Flag (CEF) should be implemented to record the specific cause of the bus-off event, distinguishing it from conventional hardware-induced failures.
- Inhibition of Automatic Recovery: According to standard CAN specifications, a node in the bus-off state may automatically attempt to recover to the error-active state after a certain sequence of recessive bits or through local reset configurations. However, in the case of a cybersecurity-induced bus-off, automatic recovery should be explicitly prohibited. Such a restriction prevents a 'recovery loop' in which the ECU is repeatedly exposed to and contested by a persistent attacker. Therefore, the recovery logic should verify the CEF; if the flag indicates a cybersecurity attack as the root cause, recovery remains inhibited until a secure diagnostic command or a power cycle is performed.
- The driver should be notified of the occurrence of a spoofing error, as it represents a potential cybersecurity threat to the vehicle's control systems. From a legal and regulatory standpoint, this notification is essential, as various standards and regulations may require prompt user notification in the event of a cybersecurity incident.

### *4.4. Algorithmic Implementation of the Proposed Mechanism*

To improve the clarity of the proposed mechanism, Algorithm 1 summarizes its operational logic in four phases. The procedure starts with the reception of the arbitration field, proceeds to the identification of the error type, then constructs and transmits the corresponding error frame, and finally invokes the intended security response. This phased description is intended to show how the proposed sixth error type can be integrated into the CAN error-handling flow without altering the basic transmission structure of the protocol.

In the first phase, the ECU waits for the start of a frame, identified by the SOF bit, and then collects the bits belonging to the arbitration field. Instead of making a decision on a bit-by-bit basis during reception, the proposed mechanism stores the received arbitration bits in RxBuffer. This design allows the ECU to defer the decision until the full arbitration field has been received and interpreted. As a result, the subsequent comparison is based on complete identifier and RTR information rather than on partial observations.

Once the arbitration field has been fully received, the second phase performs cybersecurity-error identification. At this point, the ECU extracts the received identifier and the RTR bit from the RxBuffer and compares them with its own assigned identifier and current transmission state. If the observed identifier matches the ECU's assigned transmit identifier while the ECU itself is not in the transmitting state, the event is treated as suspicious. The final condition is the RTR bit. When the RTR bit is dominant, the observed frame is interpreted as a data frame, and the event is classified as the proposed sixth error type, namely a cybersecurity error. Otherwise, the event is not treated as a cybersecurity error, and the node continues normal CAN operation.

**Algorithm 1** Detection and Response Algorithm for the Proposed Sixth Error Type

**Require:** CAN Bus bitstream, $Node\_ID$, $Node\_Status$
**Ensure:** Cybersecurity-aware error frame

**// Phase 1: Bit-by-Bit Arbitration Data Collection**
Wait for the start of a frame ($SOF$ bit)
Initialize $RxBuffer \leftarrow \emptyset$, $RxBufferCnt \leftarrow 0$
**while** $RxBufferCnt <$ Length of Arbitration Field **do**
    $RxBuffer[RxBufferCnt] \leftarrow$ Read current bit from CAN Bus
    $RxBufferCnt \leftarrow RxBufferCnt + 1$
**end while**

**// Phase 2: Identification and Differentiation**
**if** Arbitration Phase is complete **then**
    $Received\_ID \leftarrow$ Extract identifier from $RxBuffer$
    $RTR\_Bit \leftarrow$ Extract RTR bit from $RxBuffer$
    **if** ($Received\_ID == Node\_ID$) **and** ($Node\_Status \neq Transmitting$) **then**
        **if** $RTR\_Bit == Dominant(0)$ **then**
            **Identify** Sixth Error Type (Cybersecurity Error)

**// Phase 3: Error Frame Generation and Transmission**
            $Error\_Type\_Indicator \leftarrow$ Recessive (1)
            $Fixed\_Field \leftarrow$ Dominant (0)
            **Transmit** (Cybersecurity-aware Error Frame)
            *Note: The transmission of the malicious payload is neutralized due to the transmission of an error frame.*

**// Phase 4: Security Response and Visual Alert**
            Set $CEF \leftarrow 1$
            Transition to $Bus\_off$ state
            Trigger Visual Alert
        **end if**
    **end if**
**end if**
**return**

The third phase constructs and transmits the appropriate error frame according to the identified error type. If the event is classified as a cybersecurity error, the proposed cybersecurity-aware error frame is generated. In this case, the error-type indicator is assigned a recessive value to distinguish the event from a conventional reliability error, whereas the following fixed field remains dominant in order to preserve a clear boundary before the error delimiter. This prevents the interpretation of the bus idle status, as defined in CAN. The resulting cybersecurity-aware error frame is then transmitted at the bit immediately following the arbitration field, so that the suspicious frame is invalidated before payload delivery.

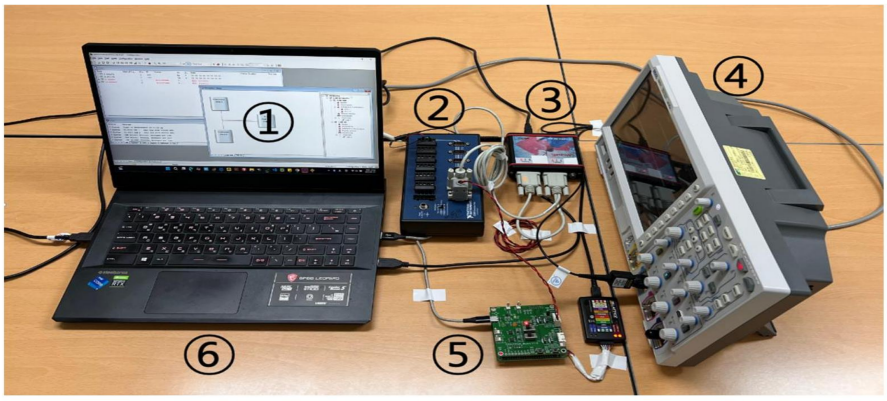
Finally, the fourth phase executes the intended follow-up response associated with a detected cybersecurity error. This phase includes updating the Cybersecurity Error Flag (CEF), forcing the node into the bus-off state, and invoking the predefined visual or HMI alert policy. These actions are not introduced as a replacement for the protocol-level mechanism itself; rather, they illustrate how the proposed sixth error type can be linked to higher-level vehicle responses after the anomaly has been detected and signaled. In this way, the algorithm provides not only a detection rule but also a structured response path that connects protocol-level signaling to subsequent vehicle-level handling.

## 5. Experimental Setup and Results

Building upon the cybersecurity-aware error frame mechanism proposed previously, this chapter presents a functional verification within a laboratory environment. The primary objective is to evaluate the effectiveness of the proposed error frame in explicitly indicating threats when a spoofing attack is detected. To this end, an experimental environment simulating a CAN-based in-vehicle network was configured.

### *5.1. Experimental Environment Setup*

The experimental environment for validating the proposed mechanism integrates both hardware and software components. Specifically, it consists of an FPGA implementing the security logic, virtual ECUs generated via Vector CANoe, a CAN breakout box, an oscilloscope, and a laptop. Figure 4 illustrates the overall configuration of this experimental environment. As shown in Figure 4a, the environment is further categorized into six primary components: (1) a laptop running Vector CANoe to host the simulation environment, (2) a CAN breakout box for physical network configuration, (3) a Vector CANcaseXL interface, (4) a RIGOL DS7014 oscilloscope for waveform verification, (5) a MachXO2 FPGA implementing the proposed security logic, and (6) a laptop computer used to host and execute the Vector CANoe environment for controlling the virtual ECUs. The specific roles, technical specifications, and Message IDs assigned to each component are summarized in Figure 4b.




**(a) Photograph of the experimental environment**

| No | Component | Role | Spec | Msg ID |
|---|---|---|---|---|
| 1 | Virtual ECU #1 | An ECU sends attack message which has same ID with virtual ECU #2 and FPGA | - | 0x100 |
| | Virtual ECU #2 | An ECU that responds appropriately after receiving an attack messages and error messages | - | 0x200 |
| 2 | CAN breakout box | A device that helps configure a physical CAN-based in-vehicle network in the Lab. environment | National Instruments 780041-01 | - |
| 3 | CAN interface | A network interface for connecting virtual ECUs to physical networks | Vector CANcase XL | - |
| 4 | Oscilloscope | A device to verify that improved error messages are actually generated according to the proposal mechanism | RIGOL DS7014 | - |
| 5 | FPGA | The victim ECU generates an error message according to the proposed mechanism | MachXO2 breakout board evaluation kit | 0x43F |
| 6 | Laptop | A PC that runs a development and testing software tool to operate the virtual ECU | - | - |

**(b) Technical specifications and roles of the experimental environment components**

**Figure 4.** Experimental environment setup. (**a**) Photograph of the experimental environment. (**b**) Technical specifications and roles of the experimental environment components.

The mechanism proposed in this study enhances the conventional error message so that it can explicitly differentiate between errors arising from ordinary faults and those triggered by malicious cyberattacks. In practical terms, enabling existing devices to represent this distinction would normally require modifications to the protocol stacks of commercial CAN controllers. However, such modifications are both cumbersome and impractical in the testing phase. Therefore, in order to circumvent the necessity of altering commercial CAN controller stacks and to provide a proof-of-concept implementation without introducing compatibility issues, the proposed mechanism was realized on an FPGA platform.

### *5.2. Experimental Scenarios and Evaluation Results*

To validate the feasibility and classification accuracy of the proposed mechanism, we conducted experiments based on two representative scenarios: (A) a conventional communication-reliability error and (B) a malicious cybersecurity attack.

To demonstrate that the extended error mechanism does not interfere with legacy CAN reliability handling, a CRC-field manipulation was performed to induce an error during an otherwise normal transmission. Specifically, Node #2 transmitted a legitimate data frame but deliberately altered the CRC bits embedded in the frame, causing the receiving nodes to observe a CRC mismatch. As a result, the receiving nodes detected a CRC error and generated a conventional CAN error frame. This legacy error frame was observed across the network, and the orange LED indicator was activated to represent the occurrence of a traditional CAN error. This indicates that even when the proposed extended mechanism coexists on certain nodes, communication-reliability errors remain properly detectable and are processed in the standard manner.

Figure 5 illustrates the FPGA board configured with LED indicators for real-time monitoring of the CAN communication state. The green LED indicates normal operating conditions, confirming that messages are transmitted and received without anomalies. The orange LED is activated when any of the five communication reliability errors—bit, stuff, CRC, form, or ACK errors—defined in ISO 11898 are detected. In contrast, the red LED signals the occurrence of cybersecurity errors, such as spoofing or replay attacks. This visual feedback system explicitly notifies both the internal ECUs and vehicle passengers of an ongoing cybersecurity attack, providing a clear distinction from traditional communication faults.

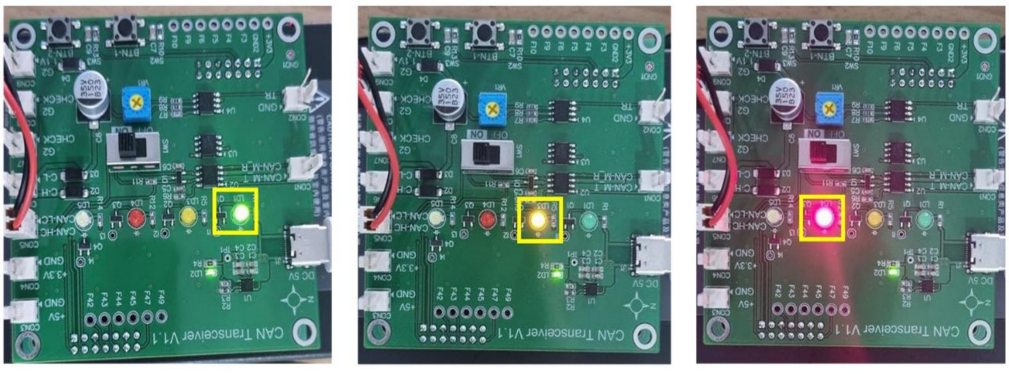



**Figure 5.** FPGA board with LED indicators.

A malicious cybersecurity attack was also executed to verify the differentiation capability of the proposed mechanism. An attacker injected a forged CAN frame bearing the victim ECU's identifier (ID: 0x43F) during the arbitration phase. Due to the broadcast nature of the CAN protocol, the victim ECU received this forged frame despite not having initiated the transmission. By comparing the observed bus traffic with its local transmit state, the victim ECU identified the inconsistency and classified the event as a spoofing attempt. Upon

detection, the security logic immediately generated the proposed error frame to invalidate the malicious transmission. As captured in the oscilloscope trace in Figure 6, although the target identifier is 0x43F, the arbitration phase is physically manifested as 14 bits. This includes the SOF bit and a dominant stuff bit inserted after the fifth consecutive recessive bit of the identifier, complying with the ISO 11898 bit-stuffing rule. Immediately following this, the proposed error frame is injected, featuring a unique two-bit cybersecurity flag where the first bit is set to recessive.

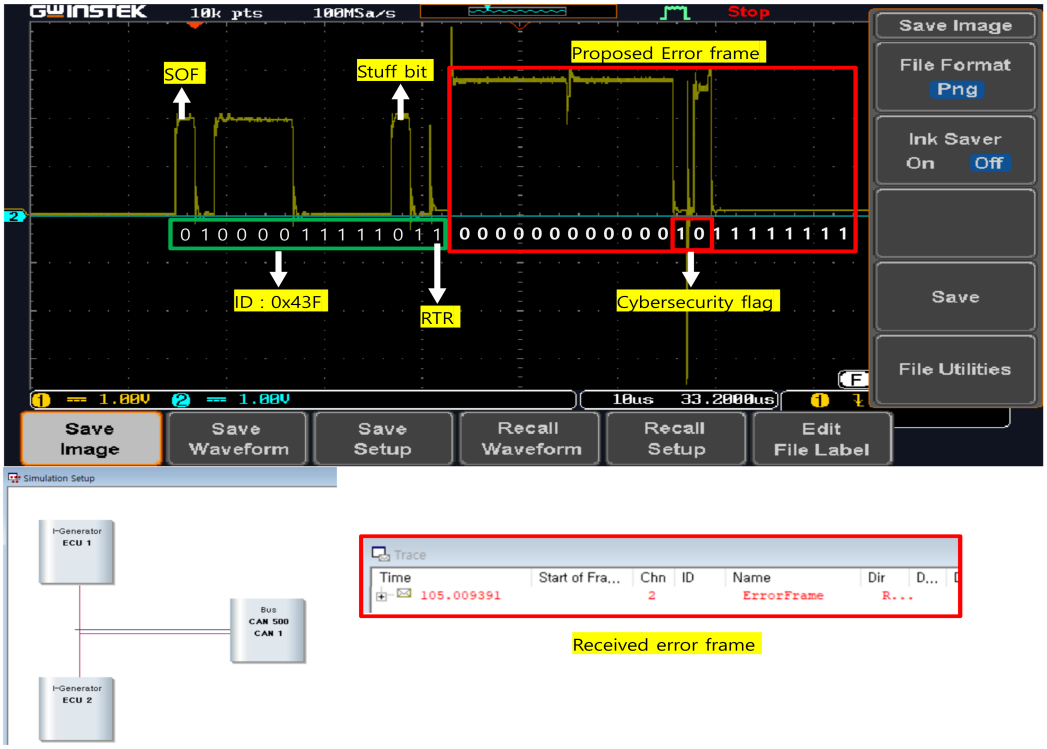



**Figure 6.** Cybersecurity Error Frames in Spoofing Attacks.

In this experiment, the attack detection latency of the proposed mechanism is observed to be 28 µs, which corresponds directly to the physical time required to receive the bits of the arbitration phase. This duration aligns with the transmission time of the 14-bit arbitration phase at a CAN communication speed of 500 kbps, given that a single bit requires 2 µs to transmit. As shown in Figure 6, the oscilloscope trace visually substantiates this 28 µs latency against a horizontal scale of 10 µs per division. Moreover, the trace reveals no temporal gap between the end of the RTR bit and the injection of the proposed error frame. This instantaneous response demonstrates that the mechanism introduces no additional processing delay at the data link layer, neutralizing the malicious payload immediately.

Additionally, the broadcast characteristics of the CAN data link layer inherently prevent false positives and false negatives within this detection logic. In CAN communication, each transmitting node is designed so that a unique identifier is assigned to each transmitted message. Because the proposed mechanism relies on identifying a clear contradiction in CAN communication, specifically when a node detects its own unique identifier on the bus while its internal transmission state remains inactive, it does not trigger false positives. Likewise, the broadcast nature of the CAN protocol ensures that any unauthorized transmission of an identifier is inevitably captured during the legitimate owner's arbitration monitoring. Thus, false negatives are structurally precluded within the evaluated scenarios.

The bottom portion of Figure 6 displays the synchronized Vector CANoe trace window, which explicitly records the transmission of an error frame on the CAN bus. This confirms that the proposed mechanism successfully invalidates the spoofed frame and that the

error is recognized across the entire network. This event activated the red LED indicator, providing an explicit alert to both internal ECUs and vehicle passengers of an ongoing cybersecurity attack.

In the experimental scenarios, spoofing attacks were performed; however, the findings are equally applicable to replay attacks due to the fundamental characteristics of the CAN protocol. During the arbitration phase, the CAN protocol transmits data bit by bit, and every node on the bus monitors each bit simultaneously. If an ECU detects its own unique identifier on the bus, it represents a structural contradiction, regardless of whether the frame was newly crafted or previously captured and replayed. Since the resulting bit sequence during the arbitration process is identical for both spoofing and replay attacks, the experimental validation of spoofing attacks serves as a comprehensive proof of concept for both attack types.

### *5.3. Comparative Analysis*

To clarify the technical contributions of our study, the proposed mechanism is compared with existing CAN bus security solutions across five key metrics, as summarized in Table 3.

**Table 3.** Comparison of the proposed mechanism with existing CAN security solutions.

| Existing CAN Security Solutions | Attack Detection Timing | Attack Detection Entity | Response Latency | Real-Time Attack Blocking | Protocol-Level Integration |
|---|---|---|---|---|---|
| Longari et al. [9] | After CAN data frame reception | External IDS ECU on CAN network | Up to 678 µs | No | No |
| Serag et al. [12] | Immediately after arbitration phase | Software agent in ECU | Up to 512 µs | Yes | No |
| Ohira et al. [13] | Before CAN data frame transmission | Internal Linux kernel module | Up to 4 s | Yes | No |
| Proposed mechanism | Immediately after arbitration phase | ID-owning ECU (self-monitoring) | Up to 28 µs | Yes | Yes |

There are differences between existing solutions and the proposed mechanism in terms of attack detection timing and attack detection entity. First, Longari et al. [9] employ an approach where an additional IDS ECU is installed on the CAN network to analyze threats after the full reception of a data frame. This is characterized as a post-event approach, as detection occurs only after the malicious message has already been transmitted and processed on the network. In contrast, Serag et al. [12] and Ohira et al. [13] utilize a separate software agent or kernel module, respectively, to identify threats and perform real-time attack blocking before the malicious message is fully delivered. While the proposed mechanism shares a similar detection timing—identifying attacks either immediately after the arbitration phase or before data transmission—it exhibits a fundamental structural departure from these methods. Specifically, the proposed mechanism enables the ID-owning ECU to self-monitor its own identifier in real-time without the intervention of external IDS devices, additional software, or kernel modules. This approach enables the implementation of real-time defense by immediately detecting contradictions in CAN communication before payload transmission begins, while simultaneously minimizing additional resource consumption.

The proposed mechanism demonstrates a low response latency of up to 28 µs. This value is lower compared to the response times reported in existing literature, such as 678 µs by Longari et al. [9], 512 µs by Serag et al. [12], and up to 4 s by Ohira et al. [13] This result stems from intercepting attacks directly at the data link layer and reduces the latency typically associated with software-based security solutions.

Furthermore, this research achieves protocol-level integration by adding a cybersecurity-specific error type to the standard error types defined in the CAN protocol. Unlike existing CAN security solutions that require external devices, additional applications, or kernel modifications, the proposed mechanism incorporates cybersecurity errors into the protocol by leveraging the error handling mechanism of the ISO 11898 standard. This approach differentiates itself from existing CAN security solutions by implementing cybersecurity capabilities that extend the standard error-handling mechanism.

## 6. Limitations

The proposed mechanism focuses on mitigating impersonation and replay attacks by taking advantage of the inherent communication characteristics of the CAN protocol. By monitoring its own unique identifier while in an inactive transmission state, a legitimate ECU can identify these identity-based threats after a 28 µs detection latency and subsequently suppress them without an additional response delay. However, the mechanism is unable to detect cases where an attacker manipulates part or all of the payload during transmission via a Man-in-the-Middle (MitM) attack after the arbitration phase has been completed. To address such malicious payload manipulation, incorporating message authentication codes (MACs) for the payload or integrating the system with higher-layer solutions like behavior-based intrusion detection systems could be beneficial.

The strategy of transitioning to a bus-off state upon detecting a cybersecurity error represents another consideration, as it could potentially be exploited by an attacker to facilitate a denial-of-service (DoS) attack. While this approach aims to isolate a compromised ECU, an attacker might intentionally spoof a valid ID to force a legitimate ECU into a bus-off state, thereby disrupting the intended operation of the ECU. Nevertheless, the proposed mechanism prioritizes passenger safety by providing visual alarms and transitioning the vehicle into a limp-home mode, ensuring a secure environment even during a cyberattack. Despite these safety-oriented measures, the structural risk of a DoS attack remains a limitation that may not be fully covered by the current mechanism, suggesting the potential need for concurrent use with behavior-based intrusion detection systems.

## 7. Conclusions

While the CAN protocol ensures high communication reliability under the ISO 11898 standard, its error-handling mechanisms are fundamentally limited to detecting communication reliability issues caused by physical faults, leaving the network structurally vulnerable to cybersecurity threats. To address this gap, this study proposes a sixth error type as a potential extension to the existing standard to support real-time detection and defense against cyberattacks in the evaluated attack scenario. This is significant in that it demonstrates feasibility in a protocol-level defense system designed to identify security threats at the data-link layer while inheriting the conventional error detection philosophy

The core of the proposed error type enables a receiving ECU to immediately classify the detection of its own identifier, paired with a dominant Remote Transmission Request (RTR) bit, as a cybersecurity error—even when it has not initiated a transmission. Upon detection, the security logic injects an error frame at the bit immediately following the arbitration field to invalidate the malicious transmission before the data payload is broadcast. For this purpose, the proposed error frame incorporates a two-bit cybersecurity flag, where the first bit distinguishes a cybersecurity-related error from a conventional communication reliability error, and the fixed dominant second bit preserves compatibility with the CAN protocol by allowing ECUs to correctly detect the bus-idle state.

The effectiveness of the proposed mechanism was demonstrated in an experimental environment through experiments using an FPGA-based prototype and a Vector CANoe

environment. Through oscilloscope waveform analysis, it was observed that during a spoofing attack, the proposed error frame is injected immediately after the 14-bit arbitration phase—which incorporates standard bit-stuffing rules—to block the attack in real time. The explicit recording of an Error Frame in the CANoe trace was demonstrated. In conclusion, this research showed that security requirements could be addressed for real-time threat detection and user notification at the protocol level through modifications of the CAN standard.

Our future research will focus on advancing active control strategies and HMI notification systems to ensure the practical safety of the vehicle in response to detected cybersecurity errors. Specifically, we plan to design efficient transition mechanisms to limp-home modes that guide the vehicle to a safe state upon error detection and develop intuitive threat-alert functions integrated with actual In-Vehicle Infotainment (IVI) systems or digital clusters.

**Author Contributions:** Conceptualization, Y.S., Y.K., Y.L. and S.W.; methodology, Y.S., Y.L. and S.W.; investigation, Y.S.; writing—original draft preparation, Y.S. and Y.L.; writing—review and editing, Y.L. and S.W.; supervision, Y.L. and S.W.; funding acquisition, Y.L. All authors have read and agreed to the published version of the manuscript.

**Funding:** This work was supported by the Technology development Program (RS-2024-00402427) funded the Korea Planning & Evaluation of Industrial Technology (KEIT, Republic of Korea).

**Institutional Review Board Statement:** Not applicable.

**Informed Consent Statement:** Not applicable.

**Data Availability Statement:** The data presented in this study are available on request from the corresponding author. The data are not publicly available due to security and proprietary restrictions related to automotive network communication.

**Acknowledgments:** During the preparation of this manuscript, the authors used Gemini 3.1 Flash (Google) for the purposes of English translation. The authors have reviewed and edited the output and take full responsibility for the content of this publication.

**Conflicts of Interest:** Yongeun Kim is employed by Synergytion. The remaining authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

## References

1. Zhu, H.; Zhou, W.; Li, Z.; Li, L.; Huang, T. Requirements-Driven Automotive Electrical/Electronic Architecture: A Survey and Prospective Trends. *IEEE Access* **2021**, *9*, 100096–100112. [CrossRef]
2. Keysight Technologies. Automotive Ethernet: Enabling the Future of Autonomous Driving. 2024. Available online: https://www.keysight.com/us/en/assets/7018-06381/white-papers/5992-3430.pdf (accessed on 13 September 2025).
3. Bozdal, M.; Samie, M.; Jennions, I. A Survey on CAN Bus Protocol: Attacks, Challenges, and Potential Solutions. In *Proceedings of the 2018 International Conference on Computing, Electronics & Communications Engineering (iCCECE), Southend, UK, 16–17 August 2018*; IEEE: Piscataway, NJ, USA, 2018; pp. 201–205. [CrossRef]
4. *ISO 11898:1993*; Road Vehicles—Interchange of Digital Information—Controller Area Network (CAN) for High-Speed Communication. International Organization for Standardization: Geneva, Switzerland, 1993.
5. *ISO 11898-1:2003*; Road Vehicles—Controller Area Network (CAN) Part 1 Data Link Layer and Physical Signalling. International Organization for Standardization: Geneva, Switzerland, 2003.
6. *ISO 11898-1:2015*; Road Vehicles—Controller Area Network (CAN) Part 1 Data Link Layer and Physical Signalling. International Organization for Standardization: Geneva, Switzerland, 2015.
7. *ISO 11898-1:2024*; Road Vehicles—Controller Area Network (CAN) Part 1 Data Link Layer and Physical Coding Sublayer. International Organization for Standardization: Geneva, Switzerland, 2024.
8. Cho, K.; Shin, K. Error handling of in-vehicle networks makes them vulnerable. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS), Vienna, Austria, 24 October 2016*; Association for Computing Machinery: New York, NY, USA, 2016; pp. 1044–1055. [CrossRef]

9. Longari, S.; Penco, M.; Carminati, M.; Zanero, S. CopyCAN: An Error-Handling Protocol based Intrusion Detection System for Controller Area Network. In *Proceedings of the ACM Workshop on Cyber-Physical Systems Security & Privacy, London, UK, 11 November 2019*; Association for Computing Machinery: New York, NY, USA, 2019; pp. 39–50. [CrossRef]
10. Bloom, G. WeepingCAN: A Stealthy CAN Bus-off Attack. In Proceedings of the 28th Annual Network and Distributed System Security Symposium (AutoSec), San Diego, CA, USA, 25 February 2021. [CrossRef]
11. Agbaje, P.; Olufowobi, H.; Hounsinou, S.; Bloom, G. From Weeping to Wailing: A Transitive Stealthy Bus-Off Attack. *IEEE Trans. Intell. Transp. Syst.* **2024**, *25*, 12066–12080. [CrossRef]
12. Serag, K.; Bhatia, R.; Faqih, A.; Ozmen, M.; Kumar, V.; Celik, Z.B.; Xu, D. ZbCAN: A Zero-Byte CAN Defense System. In Proceedings of the 32nd USENIX Security Symposium (USENIX Security 23), Anaheim, CA, USA, 9–11 August 2023; pp. 6893–6910.
13. Ohira, S.; Desta, A.K.; Arai, I.; Fujikawa, K. IVNPROTECT: Isolable and Traceable Lightweight CAN-Bus Kernel-Level Protection for Securing in-Vehicle Communication. In *Proceedings of the 9th International Conference on Information Systems Security and Privacy (ICISSP), INSTICC, Lisbon, Portugal, 22–24 February 2023*; SciTePress: Setúbal, Portugal, 2023; pp. 17–28. [CrossRef]
14. CAN in Automation (CiA). CAN XL (Extended Data-Field Length). 2025. Available online: https://www.can-cia.org/can-knowledge/can-xl (accessed on 13 September 2025).
15. Decker, P. Security concepts with CAN XL. In Proceedings of the 18th International CAN Conference (iCC 2024), Baden-Baden, Germany, 14–15 May 2024; pp. 77–83.
16. Fraunhofer IPMS; CAST, Inc. CANsec: Security for the Third Generation of the CAN Bus. 2024. Available online: https://www.cast-inc.com/sites/default/files/pdfs/2024-10/cansec-white-paper_ipms-cast.pdf (accessed on 13 September 2025).
17. United Nations Economic Commission for Europe (UNECE). *UN Regulation No. 155—Cybersecurity and Cybersecurity Management System*; United Nations Economic Commission for Europe (UNECE): Geneva, Switzerland, 2021. Available online: https://unece.org/sites/default/files/2023-02/R155e%20%282%29.pdf (accessed on 13 September 2025).
18. *ISO 26262-3:2018*; Road Vehicles—Functional Safety Part 3: Concept Phase. International Organization for Standardization: Geneva, Switzerland, 2018.
19. *ISO 21448:2022*; Road Vehicles—Safety of the Intended Functionality. International Organization for Standardization: Geneva, Switzerland, 2022.
20. *ISO/SAE 21434:2021*; Road Vehicles—Cybersecurity Engineering. International Organization for Standardization: Geneva, Switzerland, 2021.
21. Robert Bosch GmbH. *CAN Specification*, Version 2.0; Technical report; Robert Bosch GmbH: Stuttgart, Germany, 1991.
22. Grassi, P.A.; Garcia, M.E.; Fenton, J.L. *NIST Special Publication 800-63-3: Digital Identity Guidelines*; Technical report; National Institute of Standards and Technology (NIST): Gaithersburg, MD, USA, 2017. [CrossRef]
23. Lee, J.; Woo, S.; Lee, Y. A Practical Method for Identifying ECUs Using Differential Voltage. *IEEE Access* **2024**, *12*, 135028–135039. [CrossRef]