Hindawi
Security and Communication Networks
Volume 2021, Article ID 9951777, 26 pages
https://doi.org/10.1155/2021/9951777

Research Article

BusCount: A Provable Replay Protection Solution for Automotive CAN Networks

Daniel Zelle and Sigrid Gürgens

Fraunhofer Institute for Secure Information Technology, Darmstadt, Germany

Correspondence should be addressed to Daniel Zelle; daniel.zelle@sit.fraunhofer.de

Received 15 March 2021; Revised 26 July 2021; Accepted 21 August 2021; Published 30 November 2021

Academic Editor: Anjia Yang

Information technology has become eminent in the development of modern cars. More than 50 Electronic Control Units (ECUs) realize vehicular functions in hardware and software, ranging from engine control and infotainment to future autonomous driving systems. Not only do the connections to the outside world pose new threats, but also the in-vehicle communication between ECUs, realized by bus systems such as Controller Area Network (CAN), needs to be protected against manipulation and replay of messages. Multiple countermeasures were presented in the past making use of Message Authentication Codes and time stamps and message counters, respectively, to provide message freshness, most prominently AUTOSAR's Secure Onboard Communication (SecOC). In this paper, we focus on the latter ones. As one aspect of this paper, using an adequate formal model and proof, we will show that the currently considered solutions exhibit deficiencies that are hard if not impossible to overcome within the scope of the respective approaches. We further present a hardware-based approach that avoids these deficiencies and formally prove its freshness properties. In addition, we show its practicability by a hardware implementation. Finally, we evaluate our approach in comparison to counter-based solutions currently being used.

1. Introduction

Information technology has become an integral part of modern vehicles. More than 50 interconnected Electronic Control Units (ECUs) realize vehicular functions in hardware and software ranging from engine control and connected infotainment systems to future autonomous driving systems. The in-vehicle communication between ECUs is realized with bus systems like CAN (Controller Area Network Bus [1]). Further, vehicles communicate with the outside world (e.g. with their manufacturer's backend systems, with the garage's On-Board-Diagnose (OBD) devices) via different communication interfaces. Usually, these interfaces are not strictly separated from the in-vehicle network (the OBD port for example must have access to a car's ECUs to extract error codes). This poses serious security threats, one of the possible attack vectors being in-vehicle communication. The vehicle owner can for example install a tuning box to suppress or inject messages that control engine operations in order to achieve more horsepower. This in turn may damage the engine and violate the warranty. Moreover, third-party devices connected to the OBD port can inject messages to the regular in-vehicle network. In [2], Koscher et al. have shown various attack techniques like *Packet Sniffing and Targeted Probing*, *Fuzzing*, and *Reverse-Engineering*.

Multiple countermeasures were presented in the past to protect in-vehicle networks (see Section 2.4). Early work can be traced back to EVITA [3] that introduced Message Authentication Code (MAC) truncation in order to cope with the small bandwidth of field buses such as CAN. This approach has been adapted by AUTomotive Open System ARchitecture AUTOSAR in SecOC [4]. Including a freshness value in a message's MAC can in principle prohibit fuzzing or replay attacks. Most of the current approaches consider a monotonic counter value.

In this paper, we discuss our new counter-based approach BusCount based on our ideas introduced in [5],

present its full formal verification, discuss its implementation and provide a practical evaluation. We further oppose it to a generic system that captures the principles of today's counter-based approaches for freshness protection. The principle idea of our approach is to use the messages that are sent on a specific bus as a pulse generator for the counter of this bus, resulting in only one counter per bus. To cope with the loss of counter values e.g. caused by technical problems or an attack, our approach includes counter synchronization. Further, it requires the sending and reception of messages to be processed simultaneously to MAC generation and verification. Therefore, we propose a hardware-based solution: The CAN controller is enhanced by the functionality to maintain a counter and to manage MAC generation and verification while the main ECU processors can be inactive at times.

In the next section, we present the principles of in-vehicle communication based on CAN, our attack model, the protection goals we will address, current work concerning the security of CAN-based communication, and finally the characteristics of the counter-based approaches currently being discussed. Section 3 then describes the details of our approach BusCount. The following Section 4 briefly introduces our Security Modeling Framework SeMF that is then used in Sections 5 and 6, respectively, to formally model and verify both the generic counter approach and our bus counter approach. Based on these results, in Section 7.1 we present a comparison of the security aspects of both approaches while Section 7.2 introduces our proof of concept implementation showing its practicability and design decisions that substantiate our formal proof. Finally, we conclude with Section 8.

2. Principles of In-Vehicle Communication

CAN bus is the core technology for onboard communication in vehicles. Brakes, acceleration, and many further essential features are controlled by ECUs that communicate using CAN bus messages. An overview of different network structures is given in [6]. The CAN network is accessible via the OBD port allowing repair shops to access the car network. Modern vehicles also have connections through infotainment systems as well as telematic control units (TCUs) connecting the CAN bus to the outside world. By connecting the in-vehicle communication with the outside world, the necessity arose to protect its messages against malicious entities.

In this section, we describe the basics of CAN bus communication, the attacker model we take as a basis, the most relevant protection goals, and the current approaches for protecting these goals.

2.1. Basics of CAN Communication in Vehicles. The CAN bus, specified in [1], is a field bus where each entity connected to it is able to send messages and listen to every message sent on the bus. The maximum transfer rate of the highspeed-CAN is 1 Mbit/s.

A standard CAN message consists of 7 segments (Figure 1): "Start of frame" bit, a message identifier, a control field, a data field, a checksum, a confirmation field, and an "end of frame" sequence.

The 11 bit identifier which is the second section of a CAN message also represents the message's priority which is used to handle collisions. The CAN bus uses Carrier Sense Multiple Access/Collision Resolution (CSMA/CR) to prevent collisions: All ECUs start sending a CAN message simultaneously and monitor its identifier while sending. In case a dominant 0 overwrites a 1 the ECU with the lower priority stops its transmission, thereby avoiding collisions.

During the transmission of the message every ECU calculates the CRC (cyclic redundancy check) over the message and checks the correctness of it as soon as it gets transmitted by the sender. In case of a problem (e.g. if the CRC check has failed) an ECU interrupts a transmission with an error frame that invalidates the message for all receivers. Furthermore, an error counter is increased by 1 for every receiver and by 8 for the sender. Every successfully transmitted message decrements the counter. If a counter reaches 128, the ECU disables its CAN connection. This mechanism ensures that damaged ECUs do not block the entire bus communication.

Successors of CAN, like CAN FD or CAN XL, differ mainly in the frequency of transmitting data payload. CAN FD can transmit up to 64 bytes while CAN XL can handle 2048 bytes.

2.2. Our Attack Model. Attacks on in-vehicle communication have been presented first by Kocher et al. in [2]. These attacks concern manipulation of brake control and vehicle acceleration via CAN Bus by message injection. In the real world, attacks on vehicle networks have been observed that manipulate in-vehicle communication by attaching devices to the bus system, like tuning devices, AdBlue emulators [7], unauthorized OBD dongles [8], etc.

In our attack model, an attacker can send arbitrary messages on any bus she has access to. Moreover, she can overhear and record all communication on a bus she is connected to and replay all recorded messages. Finally, an attacker is able to flip bits of messages or send an error frame. This enables her to invalidate messages for other ECUs after having recorded them herself. In our scenario, an attacker does not have access to any cryptographic keys. This also includes that the attacker cannot manipulate ECUs by e.g. corrupting firmware which in turn implies that legitimate ECUs always act correctly. Furthermore, the attacker is not able to manipulate processes or storage of ECUs by physical attacks. In addition, the attacker is not able to produce a counter overflow as a sufficient counter length is chosen to prohibit this sort of attacks.

2.3. Protection Goals. A secure CAN bus communication in a vehicle needs to fulfill a set of requirements to prevent previously introduced attacks. These include e.g. data integrity, confidentiality, and availability. In the following, we explain those requirements we specifically address in this paper.

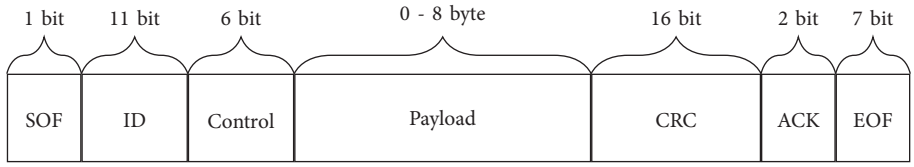

Figure 1: A single CAN frame.

Data Origin Authenticity: A message in a vehicle network should be accepted if and only if it has authentically for the recipient been sent by another valid member of the network. This property prevents attackers from manipulating messages or sending messages on the bus system from additional devices or replaced components without the intended recipient noticing.

Immediacy: Contrary to many other IT-systems, it is important for an automotive system to receive and process messages within a certain time frame. Once this time frame has passed, messages might be authentic, but can still cause fatal results, e.g. if breaking signals sent by the anti-lock braking system are processed too late. Immediacy expresses the fact that a message sent at time t_1 is accepted until t_2 if and only if $t_2 - t_1$ does not exceed a specified limit.

Non-repeatability: The last important property for an automotive network is non-repeatability: If a message is accepted at time t_1, the same message is not accepted at any later point in time. Thus, an attacker cannot eavesdrop on a message and successfully replay it at a later point in time.

Many articles do not distinguish between the two above properties and use the more abstract concept of "message freshness" with "message replay" seeking to violate freshness. We adapt to this notation and distinguish the specific characteristics when necessary.

2.4. State of the Art CAN Bus Security. The security of bus communication in current vehicle networks has already been discussed in literature and standardization. Early work on MAC truncation for secure CAN bus communication can be traced back to [3]. In this section, we give an overview of state of the art with a focus on replay protection in CAN bus systems and compare the techniques.

A lot of CAN bus security approaches introduce message authentication mechanisms, but not all introduce replay protection. The latest example of an approach without freshness values is TOUCAN: A proTocol tO secUre Controller Area Network presented by Bella et al. [9], which introduces a 24 bit truncated Chaskey MAC and a SPECK64 encryption for each CAN message.

A more exotic approach for replay protection is used in LCAP by Hazem et al. [10]. LCAP appends a truncated element of a hash chain to the CAN message and encrypts the resulting message. An HMAC secures the transmission of the last element of the hash chain to initialize the communication. Woo et al. [11] periodically change HMAC keys to prevent replay attacks.

Nürnberger and Rossow [12] developed VatiCAN, an HMAC based authentication procedure that sends a MAC in a separate message following the original CAN message. The MAC is then validated with a delay of about 4 ms. Replay protection is implemented with a monotonically incremented counter, its starting value being a random nonce generated by a central component for every message ID. The authors recommend this procedure only for a few CAN messages since it increases the bus load. Van Bulck et al. improved this approach in [13] by introducing software isolation and attestation as well as key update mechanisms.

Hartkopp et al. presented a further approach to introduce freshness to CAN messages. MaCAN [14] formally verified in [15] introduces a central trusted time server which distributes time information over the network. This information is used as freshness value for message authentication.

AUTOSAR specifies the protection of communication in vehicle networks based on a MAC and a freshness value. The specification of the Secure Onboard Communication (SecOC) [4] module suggests to add a truncated timestamp or message counter and a truncated authenticator to every message. The specific counter mechanism is based on splitting the counter (with a maximal length of 96 bits) into three different parts: the so-called "trip counter" that only changes essentially with every new trip, a "reset counter" that is reset periodically, and the actual "message counter". Only the trip counter is stored in non-volatile memory, thus mitigating loss of counter values in case of sudden ECU shutdown. The truncated freshness value has a length between 0 and 8 bit. The truncated authenticator consists of the first 24 to 28 bits of the MAC covering the full freshness value and the message.

Similar to SecOC many approaches in literature use counters and an application-level protocol to ensure replay protection. Kurachi et al. [16] suggest attaching a truncated MAC (8 bit) and a truncated monotonic counter (4 bit) to a message. A monitoring node verifies messages during transmission and overwrites invalid messages with an error frame. ECUs do not verify messages. Groll et al. [17] suggest an initialization phase to form groups of ECUs. These groups generate a shared symmetric key using an asymmetric key exchange. Within these groups, ECUs use the shared secret for authentic and confidential encryption. A counter should be part of the message to protect against replay attacks. Lin et al. presented an approach in [18] with symmetric keys for message authentication. A sender calculates a MAC for every receiver. Every ECU also holds two counters for replay protection per message ID, the last counter it has sent and the last one it has received. Every receiver can verify the MAC and process its corresponding message. The LeiA protocol by Radu et al. [19] is another solution that transfers MAC and counter value in a separate message. Every ECU has a session key for each relevant message ID derived from a long-term symmetric key and renewed after a certain period.

VeCure [20] is a CAN authentication framework similar to VatiCAN. The authentication value is also transmitted via a separate message, but contrary to VatiCAN the second message includes a *Node-ID* besides a *Message Counter* and a four byte HMAC value.

Alternatively, several approaches suggest the use of CAN+ [21], a protocol extension for CAN allowing to transport 120 bit additional data. The first approach is CANAuth presented by Van Herrewege et al. [22]. Another one is LiBrA-CAN [23]. LiBrA-CAN introduces (Linearly) Mixed MAC, which mixes multiple MACs of one message generated with different keys allowing receivers to verify a MAC even though they do not know all keys. The approach allows making sure receivers cannot impersonate a sender in a properly organized group. Both approaches send counter values in their messages to protect against replay attacks.

Some works are also considering the implementation of a secure CAN bus controller. Their approaches introduce the calculation of MACs, denial of service countermeasures, or intrusion prevention mechanisms. [24] implemented a CAN controller including a physical unclonable function implementation, key generation and storage, and encryption and decryption allowing authenticated communication over CAN. However, the approach does not consider replay protection. Ueda et al. presented a CAN controller with integrated HMAC in [25]. To ensure replay protection a truncated monotonic value of 4 bits is part of every message. Messages which are not authentic are destroyed while correct messages update the counter.

A new approach by Groza et al. [26] suggests replacing CAN IDs with a specific MAC-based algorithm that preserves the order of CAN IDs. In predefined time intervals the counter included in the MAC is incremented thus the IDs change. This approach increases the resistance against reverse engineering and denial of service attacks related to a specific ID. It does not provide data integrity and authenticity which needs an additional security protocol as mentioned in the paper. Moreover freshness is not guaranteed since the counter used in the MAC of the CAN ID does not change with every message. In case of constantly changing counter values (IDs) and if a significant limitation of ID range is acceptable this can be a viable alternative to transfer of fresh counter values.

We observed that most of the presented approaches (cf. Table 1) have similar ways to ensure replay protection and authentication of messages. All approaches add a MAC to a CAN message. While a MAC provides authenticity of a message, only in combination with a freshness value replay and delay attacks can be mitigated. Most approaches introduce a counter value to provide freshness since the usage of time or nonce values has disadvantages, discussed e.g. in [27]. The transmission of MAC and freshness values is either realized in an additional message or achieved by including truncated values in the same message. The verification of a complete message is performed by the receiver or an additional node. In the following section, we present a detailed generic model covering the characteristics of the current counter-based approaches for freshness. This model is then compared to our approach based on formal verifications of the security goals.

2.5. The Generic Counter Concept. Considering the recent research, we simplified the approaches in order to generate an abstract model to evaluate the security of software-based freshness techniques compared to our approach. Since a large majority of approaches favor counters for freshness values, we focus on this technique.

Figure 2 illustrates the abstract protocol we assume. To transmit a CAN message m which contains the $I\ D$ and payload data msg an ECU first calculates a MAC covering m and a local counter c_a derived by incrementing the previously used one (steps 1 and 2). For our analysis, the choice of the MAC algorithm is not important. In the next steps, m, c_a and the authentication tag are concatenated (step 3) and the values are transmitted (step 4). Note that this transmission is not necessarily processed with one CAN message only, different techniques could apply here. Finally, a receiver gets the message, verifies the MAC and tests if its local counter c_b is smaller than the counter in s. The check is not necessarily performed by the same entity which later processes m. If both checks are successful, the local counter is set to the counter in s and the message can be processed. Otherwise, the message is discarded.

Only some approaches consider an explicit synchronization of counter values which is necessary in case an ECU loses its counter value, e.g. due to a software error, a power loss (engine stop or malfunction) or an ECU without persistent memory. Most approaches that use synchronization introduce a central system sending an authenticated message containing the current counter value. In case a sender has an incorrect counter, the value needs to be provided by a dedicated entity or some client. In both cases the counter value is transmitted in the payload CAN message with a reserved ID. This message is secured identically to regular messages.

Even though the generic counter protocol is fairly simple it represents the characteristic properties of all above mentioned protocols that increment counters after successful validation of the message. The fact that the local counters of message recipients only change when a message is accepted is a very important characteristic property. Consequently, these protocols cannot prohibit so-called delay attacks, as will be formally shown in Section 5.2. For such an attack, the adversary with the abilities described in Section 2.2 stores and then invalidates a message and all subsequent ones related to the same counter. The reinserted message will then be accepted by the intended recipients at any later point in time if no further countermeasures are taken.

3. BusCount: A Hardware Based Bus Counter Solution

In this section we describe BusCount, our approach for a hardware based secure CAN bus communication, which eliminates the possibilities of attackers with the abilities presented in Section 2.2. We first introduce our approach to ensure immediacy, non-repeatability, and authenticity of messages on the bus and then elaborate on the synchronization mechanism for freshness values.

Table 1: Comparison of different authentication approaches for CAN-Bus (HW: Hardware, SW: Software, C : Counter, T : Timestamp, N : Nonce, *: not described)

	HW change	SW change	Central component	Freshness technique	MAC	Encryption	Transfer techniques for MAC	Syncronisation of freshness value
AUTOSAR [2]	-	✓	-	C / T	✓	-	28 bit data field	✓
CaCAN [24]	✓	✓	✓	C	✓	-	8 bit data field	-
CANAuth [35]	✓	✓	-	C	✓	-	CAN+	(✓)
Groll et al. [14]	-	✓	✓	C	✓	✓	*	-
LeiA [31]	-	✓	-	C	✓	-	sep. message	✓
LibrA-CAN [16]	-	✓	✓	C	✓	-	CAN+	-
Lin et al. [25]	-	✓	-	C	✓	-	*	(✓)
Ueda et al. [34]	✓	✓	✓	C	✓	-	8 bit data field	-
VeCure [37]	-	✓	-	C	✓	-	separate message	-
MaCAN [20]	-	✓	✓	T	✓	-	32 bit data field	✓
VatiCAN [28]	-	✓	✓	C + N	✓	-	separate message	✓
vulCAN [7]	✓	✓	✓	C + N	✓	-	separate message	(✓)
Woo et al. [38]	-	(✓)	(✓)	key refresh	✓	✓	CAN-FD	-
LCAP [21]	-	✓	-	hash chain	-	✓	16 bit extended ID	✓
TouCAN [3]	-	✓	-	-	✓	✓	24 bit data field	-
Siddiqui et al. [33]	✓	✓	✓	-	✓	✓	data field	-
CAN-TORO [15]	✓	✓	✓	Authenticated ID	-	-	-	(✓)

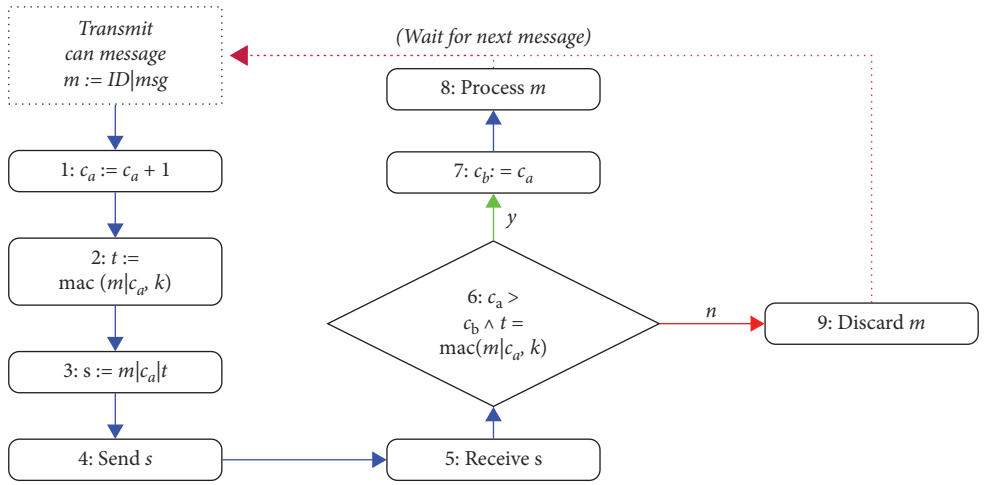

Figure 2: Process of Generic Counter Communication.

In a bus system, each participant can see and thus count every message written to the bus. Hence, the number of messages sent on a particular bus is an inherent part of the system that can serve as a bus specific counter [5] known by all devices connected to it. Consequently, there is no need to send counters. Each bus of a vehicle system is equipped with

its own counter. Its value changes automatically with each new message: An ECU sending/receiving a message reduces its local counter value by 1 and authenticates/verifies the message including the counter with a MAC. This idea can be also applied to any other bus network.

The procedure of BusCount is described in more detail as follows (see Figure 3 for a schematic representation):

First, a sender ECU starts transmitting a message m which is composed of a message ID and the payload msg. As soon as the transmission starts, the local counter c_a of the sender is temporarily decremented (step 1.1). The counter is decremented instead of the usual incrementation because 0 is the dominant bit on the CAN bus thus a lower counter can overwrite a larger counter. This property is used for the synchronization of the counter explained in the next section. A receiver ECU, when receiving the start of the message, decrements its local counter c_b and uses the result as its new counter value (step 2.1). Since sending and receiving of messages is processed simultaneously and thus the sender also receives its own message, it decrements its counter analogously. After the counters are decremented the sender starts sending mgs and both receiver and sender start calculating the MAC over the message m and their respective local counter using a shared key k. Finally, the tag t_a of the sender's MAC is transmitted (step 1.3) and received (step 2.3). All ECUs now evaluate the tag.

If the evaluation is positive m can be processed. Otherwise, the receivers whose verification failed immediately transmit an error frame which has the effect that m is discarded by every ECU. Note that this effects also ECUs that do not implement the protocol: They will discard messages overwritten with an error frame. In case multiple errors occur, a synchronization is necessary.

3.1. Synchronization. Multiple transmission failures may indicate that an ECU is not synchronized. This situation can occur e.g. if the ECU is switched off without having been able to store the current correct counter value in persistent storage. Consequently, a synchronization between all entities of a bus is necessary. Our synchronization concept utilizes the mechanism used for collision resolving which is based on the fact that sending a 0 always overwrites a 1 on the CAN bus. Hence, in BusCount counters are decremented instead of the usual incrementation.

The mechanism is illustrated in Figure 4. One ECU initializes the synchronization by sending a predefined synchronization ID (Step 1.1). At the same time, each receiving ECU including the initiator of the synchronization, receiving the start of message bit, decrements its local counter. Now all ECUs, having identified the message as synchronization message, simultaneously start to send their respective newly decremented local counter in the data frame (Step 2.1 and 2.2). The lowest counter will overwrite larger counters and ECUs with larger counters stop sending (Step 3). An ECU with the lowest counter value (the actual sender of the synchronization message) sends a MAC over the ID and the counter value (Step 4.1 or 4.2). Each ECU as a recipient of this message verifies the

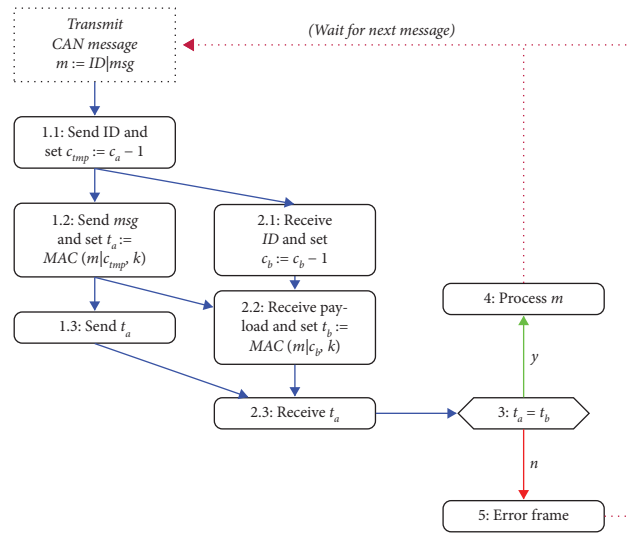

Figure 3: Process of CAN message transmission of BusCount.

correctness of the MAC and compares the counter to its local one. In case one check by any of the receivers fails, the rest of the message is overwritten with an error frame and is discarded by all controllers. Otherwise, every receiver replaces its local counter with the counter of the synchronization message.

In case multiple ECUs have the lowest counter in the network, each sends the same message without noticing each other. This concept has been used for example in [28] to implement a key exchange on a CAN bus.

In contrast to other approaches that use a counter for every message ID, our approach allows to synchronize all participants of a bus communication with just one message. Further, no central entity is needed for the process, any ECU connected to the bus can initialize it. The only condition for it to work is that at least one ECU owns and processes the correct counter value.

In Section 7.2 we will discuss design decisions and introduce our proof of concept implementation. In the next section we will briefly introduce the Security Modeling Framework SeMF that is then used in Sections 5 and 6, respectively, to formally model and verify both the generic counter approach and our bus counter approach with respect to the desired security properties. The achieved results will then be discussed in Section 7.1.

4. The Security Modeling Framework SeMF

We use our Security Modeling Framework SeMF (see [29] for a detailed description) to formally model and verify the two counter systems discussed in this paper. SeMF is a powerful modeling framework that we have already successfully applied to a variety of different domains and abstraction levels. For example, we used it to verify that specific security properties of service based systems are preserved under composition [30]. We also applied it to model and verify the integration of device authentication based on TPM attestation with secure channel establishment via SSL [31]. Another example is [32] where we proved preservation of

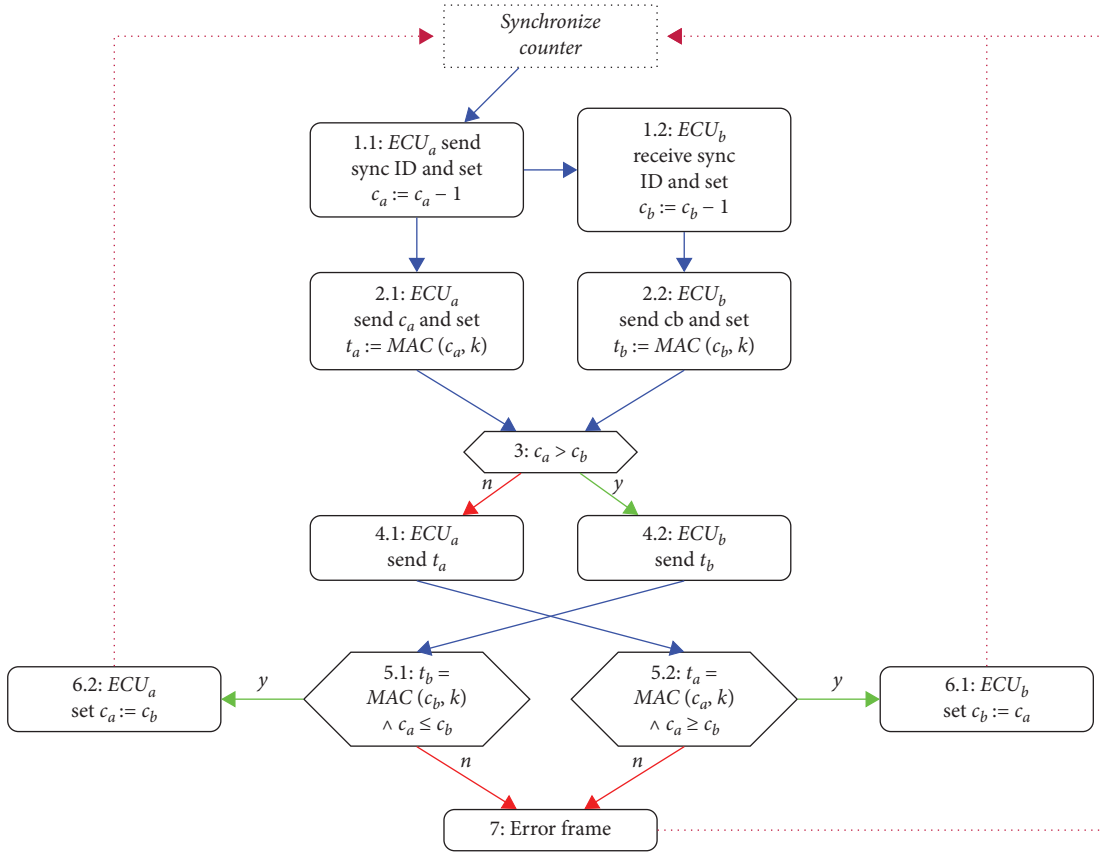

Figure 4: Synchronization of BusCount.

specific security properties for the composition of abstract security patterns.

The basic idea of SeMF is to describe the system behavior by sequences of actions that capture essential changes in the system. As underlying formal semantics SeMF uses prefix closed formal languages (see e.g. [33]) whose alphabet is composed of the actions in the system. More specifically, it uses a set of agents $\mathbb{P}$ (where the term "agent" can denote any entity acting in the system such as a human being, an ECU, etc.), and a set of actions Σ (e.g. specifying sending and receiving messages on a bus) performed by the agents. The system's behavior is then formally described by a prefix closed formal language $B \subseteq \Sigma^*$ (Σ^* denoting the set of all words composed of elements in Σ with $\varepsilon \in \Sigma^*$ denoting the empty word), i.e. by the set of its possible sequences of actions. A system model further comprises the agents' local views (denoted by λ_P for agent P). The local view of different agents usually differs since it describes which parts of the system behavior the agents can actually see (an ECU for example may see its own internal actions, but not those of other ECUs). A system model finally includes the so-called agents' "initial knowledge" $W_P \subseteq \Sigma^*$ which is defined to be prefix closed and to contain B. This concept is used in order to specify system constraints and assumptions.

Security properties are defined in terms of the system specification. The underlying formal semantics then allows formal validation, i.e. allows proving that a specific formal model satisfies specific security properties.

The following notations are used: For $\Upsilon \subseteq \Sigma^*$ and $\omega \in \Upsilon$, $\omega^{-1}(\Upsilon)$ denotes the set of all continuations of ω in Υ. For $\Gamma \subseteq \Sigma$ and $\omega \in \Sigma^*$, $\mathrm{card}(\Gamma, \omega)$ denotes the number of occurrences of any action of Γ in ω, $\mathrm{alph}(\omega)$ denotes its alphabet (i.e. the set of its actions), $\mathrm{pre}(\omega)$ is its set of prefixes, $\mathrm{pre}_1(\omega)$ denotes its first and $\mathrm{suf}_1(\omega)$ its last action. For $\omega = x_1 \ldots x_k \in \Sigma^*$ and $i \in \{1, \ldots, k\}$, $\mathrm{prevact}(x_i, P, \omega)$ denotes the last action before x_i in ω performed by agent P (in case x_i is P 's first action, $\mathrm{prevact}(x_i, P, \omega) = \varepsilon$). For $\omega \in \Sigma^*$, the function $\mathrm{actCnt}: \Sigma \times \Sigma^* \longrightarrow \mathbb{N}s$ enumerates strictly monotonically increasing the actions of ω in their order of occurrence: $\mathrm{actCnt}(a, \varepsilon) := 0$ for all $a \in \Sigma$, $\mathrm{act\,Cnt}(a, \omega) := 1$ for $\omega = a$, and for $\mathrm{card}(\Sigma, \omega) = k > 1$ we define $\mathrm{act\,Cnt}(\mathrm{suf}_1(\omega), \omega) := \mathrm{act\,Cnt}(\mathrm{suf}_1(\mathrm{pre}_{k-1}(\omega)), \mathrm{pre}_{k-1}(\omega)) + 1$.

We extend SeMF by a formal specification of actions and a homomorphism to extract any parameter of an action:

Definition 1 (Set of actions). *Let* $\mathtt{P} = \{\mathrm{par}_1, \ldots, \mathrm{par}_n\}$ *a set of parameters* ($n \in \mathbb{N}$) *and for* $j \in \{1, \ldots, n\}$ *let* V_j *the set of possible values of* par_j *with* $V_1 := \mathtt{A}$ *a set of action names and* $V_2 := \mathbb{P}$ *a set of agents. Then the set* Σ *of actions of a system* $\mathbb{S}$ *can be defined as follows:*

$$\Sigma \subseteq \bigcup_{\mathrm{par}_{i_1} \in \mathtt{A}, \mathrm{par}_{i_2} \in \mathbb{P}, \{i_3, \ldots, i_k\} \subseteq \{3, \ldots, n\}} \left(\mathrm{par}_{i_1}, \ldots, \mathrm{par}_{i_k}\right). \quad (1)$$

The sending of a message on a CAN bus can for example be formalized by (send, ECU, bus, msg). See Sections 5.1 and 6.1 for the concrete sets of actions of the two models

introduced in this paper. In order to express relations between parameters of different actions, we need to extract them from the actions:

Definition 2 (Parameter extraction). *Let* Σ *be defined as in Definition* 1. *We define a homomorphism* $\widehat{\kappa}_{\mathrm{par}_i}\colon \Sigma \longrightarrow \mathrm{P} \cup \Sigma$ *by*

$$\widehat{\kappa}_{\mathrm{par}_i}((\mathrm{par}_1, \ldots, \mathrm{par}_k)) \coloneqq \begin{cases} \mathrm{par}_i, & \text{if } \mathrm{par}_i = \mathrm{par}_i, \\ (\mathrm{par}_1, \ldots, \mathrm{par}_k), & \text{else.} \end{cases} \tag{2}$$

The security property provided by a MAC mechanism can be formally specified by the concept of authenticity introduced in [34] : A set of actions Γ is authentic for agent P after a sequence ω of actions has happened if in all sequences that P considers possible after ω, some time in the past an action in Γ must have happened. Formally:

Definition 3 (Authenticity). *A set of actions* $\Gamma \subseteq \Sigma$ *is authentic for* $P \in \mathbb{P}$ *after a sequence of actions* $\omega \in S$ *with respect to* W_P *if* $\mathrm{alph}(x) \cap \Gamma \neq \varnothing$ *for all* $x \in \lambda_P^{-1}(\lambda_P(\omega)) \cap W_P$.

The following weaker property describes that in all sequences of a language L that contain a specific action b, this action is preceded by one of the actions contained in $\Gamma \subseteq \Sigma$:

Definition 4 (Precedence). *For* $L \subseteq \Sigma^*, \Gamma \subseteq \Sigma, b \in \Sigma$ *the property* $\mathrm{prec}_L(\Gamma, b)$ *holds if for all* $\omega \in \mathrm{pre}(L)$ *with* $b \in \mathrm{alph}(\omega)$ *follows* $\Gamma \cap \mathrm{alph}(\omega) \neq \varnothing$. *We simply write* $\mathrm{prec}(\Gamma, b)$ *if from the context the language referred to is clear.*

Additionally to authenticity, we require the counter systems to provide immediacy and non-repeatability. In order to define a respective security property within SeMF we introduce the concept of a *phase class* that allows modeling that a particular action occurred within a particular period of the system. We base our definition on the concept of a phase introduced in [35]. Here a subset of Σ^* is a phase for B if it is a prefix closed language consisting only of words which, as long as they are not maximal, show the same continuation behavior within the phase as within B. Our definition transforms this to arbitrary subsets of Σ^*, not requiring them to be prefix closed:

Definition 5 (Phase class). *Let* $\Upsilon \subseteq \Sigma^*$. *A language* $\Phi(\Upsilon) \subseteq \Sigma^*$ *is a phase class for* Υ *if the following holds:*

1. $\Phi(\Upsilon) \cap \Sigma \neq \varnothing$
2. $\forall \omega, u \in \Upsilon$ *with* $\omega = uv$ *and* $v \in \Phi(\Upsilon) \setminus (\max(\Phi(\Upsilon)) \cup \{\varepsilon\})$ *holds:* $\omega^{-1}(\Upsilon) \cap \Sigma \subseteq v^{-1}(\Phi(\Upsilon)) \cap \Sigma$

Thus, a phase class is characterized by being closed with respect to concatenation. Maximal words in a phase class, denoted by $\max(\Phi(\Upsilon))$, are those $v \in \Phi(\Upsilon)$ for which holds $va \notin \Phi(\Upsilon)$ for all $a \in \Sigma$ (i.e. no matter whether or not exists $\omega = uva \in \Upsilon$).

A phase class can be a very complex construct. However, in many cases phase classes are of interest that can be defined by the actions that start and terminate, respectively, the words. The following definition takes into account that an action can occur more than once in a word. Each starting action occurring in a word $\omega \in \Upsilon$ starts a word of Φ. The word ends with the first j_i occurrences of an action in T_i:

Definition 6 ((S,T)-phase class). *Let* $\Upsilon \subseteq \Sigma^*$, $S \subseteq \Sigma$, $T = T_1 \cup \ldots \cup T_k \subseteq \Sigma$ $(k \in \mathbb{N})$ *with* $T_i \cap T_j = \varnothing$ *for all* $i \neq j$. *Then* , $\Phi \coloneqq \Phi(\Upsilon, S, \{(T_1, j_1) \ldots, (T_k, j_k)\}) \subseteq \Sigma^*$ *is a phase class for* Υ *starting with* S *and terminating with respect to* $\{(T_1, j_1), \ldots, (T_k, j_k)\}$ *if*

1 Φ *is a phase class for* Υ,

2 $\Phi(\Upsilon) \cap \Sigma = S$

3 *for all* v *maximal in* Φ *the following holds: For* $\omega, u \in \Upsilon, z \in \Sigma^*$ *with* $\omega = uvz$ *it follows* $z = \varepsilon$ *or there exists* $i \in \{1, \ldots, k\}$ *such that* $\mathrm{suf}_1(v) \in T_i$, $\mathrm{card}(T_i, v) = j_i$, *and* $\mathrm{card}(T_l, v) < j_l$ *for all* $l \in \{1, \ldots, i-1, i+1, \ldots, k\}$.

We call such a phase class an $(S, \{(T_1, j_1), \ldots, (T_k, j_k)\})$ -phase class for Υ. If all words in the phase class terminate with the first occurrence of any $t \in T$, we simply call it an (S, T) -phase-class for Υ, denoted by $\Phi(\Upsilon, S, T)$.

It can easily be shown that an (S, T) -phase class for a prefix closed language is itself prefix closed. (S, T) -phase classes are a very useful concept for the concrete specification of freshness properties. We can further combine these two concepts with authenticity:

Definition 7 (Authenticity within a phase class). *Let* $B \subseteq \Sigma^*$ *be the behavior of a system,* $\omega \in B, b \in \mathrm{alph}(\omega)$, *and* $\Phi(W_P) \subseteq \Sigma^*$ *a phase class for agent P's initial knowledge* W_P. *A set of actions* $\Gamma \subseteq \Sigma$ *is authentic for P after* ω *within* $\Phi(W_P)$ *and with respect to* λ_P *and b if (i) it is authentic for P after* ω *and if (ii) for all* $x \in \lambda_P^{-1}(\lambda_P(\omega)) \cap W_P$ *for which exists* $u, z \in \Sigma^*$ *and* $v \in \Phi(W_P)$ *such that* $x = uvz$ *and* $b \in \mathrm{alph}(v)$ *it follows* $\mathrm{alph}(v) \cap \Gamma \neq \varnothing$. *If the property holds for all* $\omega \in B$, *we denote this property shortly by* $\mathrm{authWiPhase}(\Gamma, b, P, \Phi(W_P))$.

5. Formalization and Verification of the Generic Counter Approach

In this section, we introduce our SeMF model of the generic counter system described in Section 2.5 (denoted by GenCnt henceforth) and formally prove to which extend it satisfies the protection goals data origin authenticity, immediacy and non-repeatability of messages.

5.1. The Formal Generic Counter Model. Our SeMF model shall be as simple as possible. It needs to include ECUs connected to a bus whose messages shall be proven to be protected. It also needs to reflect our attack model introduced in Section 2.2, hence must include devices (e.g. ECUs) that an attacker can use to monitor, record, resent and manipulate messages sent on the bus. It is obvious that messages sent on one bus may be accepted by ECUs connected to another bus if the key used for protecting these messages is the same for both buses. Hence we disregard this aspect and restrict our model to one group of honest devices,

all owning the same MAC key for message protection, and a further device Eve representing dishonest behavior. All devices are connected to the same bus. Our model can easily be extended, for example by adding more groups, keys and buses, in case other aspects than those addressed in this paper shall be investigated. A special honest device is the Fresh Value Master (FvM) that is responsible for the synchronization of ECUs regarding their counter. We assume all honest devices to act according to a given specification (see Section 5.1.3). FvM only sends synchronization messages, i.e. messages with msgid = `sync`. Other honest devices receive synchronization messages and send and receive functional messages with msgid = `fmsg`. We do not distinguish between different types of functional messages and use just one with its corresponding message counter. Eve can send and receive all types of messages but does not own the MAC key and can thus not generate MACs.

We use four different types of actions: sending and receiving (i.e. accepting) of messages, reading a message without processing it, and an action that models an ECU losing the correct message counter (denoted by genCnt henceforth). This action comprises any situation in which an ECU is not synchronized anymore, i.e. owns a counter smaller than the current correct counter value.

While in many systems (e.g. in [4]) messages only contain a truncated message counter and MAC, respectively, in our model the counter's complete value is included. This way we model the assumption that the recipients always succeed in determining the counter values used by the senders (after all, this aspect is not in the focus of our investigations).

5.1.1. Agents, Parameters and Actions. For the formal specification of actions according to Definition 1, we use the following sets:

(1) Set of agents:

$\mathbb{P}_{gCnt} := \mathscr{E}\mathscr{C}U_{gC} \cup \{\text{Eve}\}$ with $\mathscr{E}\mathscr{C}U_{gC} := \{\text{ECU}_1^{gC}, \text{ECU}_2^{gC}, \text{FvM}\}$ whose members are connected to the only bus of the system. FvM denotes the synchronization master and Eve denotes a further device being connected to the bus but not being member of $\mathscr{E}\mathscr{C}U_{gC}$.

(2) set of action names: $\mathtt{A}_{\mathtt{gCnt}} = \{\text{send}_{gC}, \text{read}_{gC}, \text{recv}_{gC}, \text{loseCnt}_{gC}\}$

(3) set of parameters:

$\mathtt{P}_{\mathtt{gCnt}} :=$ {aname, ecu, ecukey, ecucnt, prevcnt, bus, mackey, msgid, msg, cnt} with aname $\in \mathtt{A}_{\mathtt{gCnt}}$, ecu $\in \mathbb{P}_{gCnt}$, ecukey, mackey $\in \{\mathtt{key}\} \cup \mathbb{N}$, `key` being the key all honest ECUs use for MAC generation and verification, while ecukey $\in \mathbb{N}$ for ecu = Eve. Further, ecucnt, prevcnt, cnt $\in \mathbb{N} \cup$ {nocnt}, bus $\in$ {bus}, msgid $\in$ {sync, fmsg}, and msg $\in \mathscr{M}$ ($\mathscr{M}$ being an arbitrary set of messages).

(4) The set of actions $\Sigma_{g\text{Cnt}}$ is then defined as follows:

(1) (send_{gC}, ecu, ecukey, ecucnt, prevcnt, bus, mackey, msgid, msg, cnt): ecu $\in P_{gCnt}$ sends a message on bus = `bus`. The message's MAC (not explicitly modelled by a parameter of this action) is generated with mackey and covers msgid, msg and cnt. ecu $\in \mathscr{E}\mathscr{C}U_{gC}$ if none of the entire message bits as illustrated in Figure 1 has been written to the bus by Eve. ecu may or may not have generated the MAC. For ecu $\in \mathscr{E}\mathscr{C}U_{gC}$, the parameter ecukey denotes ecu 's MAC generation and verification key `key`, ecucnt denotes its local counter value after having performed the send_{gC} action, and prevcnt denotes the counter value resulting from ecu's previous action (see Section 5.1.3 for the specific operations regarding an ECU's counter). The message can be a functional message, indicated by msgid = `fmsg`, in which case the counter contained in the message's payload is explicitly modelled by cnt, or a synchronization message with msgid = `sync`. In this case the message's payload msg only contains the counter determined by the sender and the parameter cnt contains the constant `nocnt`. Note that there is the possibility that the message is altered (by a technical error or by Eve) after having been sent and may thus only cause a read_{gC} action (see below).

(2) (read_{gC}, ecu, ecukey, ecucnt, prevcnt, bus, mackey, msgid, msg, cnt) denotesecu $\in \mathbb{P}_{bCnt}$ reading a message without processing it (i.e. without accepting it). The action does not change the local message counter ecucnt if ecu $\in \mathscr{E}\mathscr{C}U_{gC}$ (see Prop.A10 below).

(3) (recv_{gC}, ecu, ecukey, ecucnt, prevcnt, bus, mackey, msgid, msg, cnt) denotes the successful reception and processing of a message by ecu $\in \mathbb{P}_{gCnt}$.

(4) With the action ($\text{loseCnt}_{\text{gC}}$, ecu, ecucnt, prevcnt, msgid) we model the fact that ecu $\in \mathscr{E}\mathscr{C}U_{gC}$ has lost the correct counter value for some reason. This action comprises any situation in which an ECU is not synchronized anymore. As a consequence, its counter is set to a value smaller than the correct counter value.

The idea of a generic counter-based system is that counter values should be strictly monotonically increasing. However, in real systems message transmission may fail due to transmission errors of the bus (e.g. by flipping a bit). Such a message is not accepted in which case the sender simply repeats it, using the same counter as before. We abstract from this since incidents of this type are not security relevant and assume all messages sent by an honest ECU not to suffer from physical failures of the system. This leads to the following definition of the correct counter for a specific action.

Definition 8. Let $\omega = x_1 \dots x_r \in \Sigma_{gCnt}^*$ *and* $k \in \{1, \dots, r\}$. *Then the correct counter for action* x_k *in* ω *is defined as follows:*

$$\operatorname{cor}\mathrm{Cnt}_{gC}(x_k,\omega) \coloneqq \begin{cases} 1, & \text{if } k=1, \\ \operatorname{card}(\{x_i \in \operatorname{alph}(\omega) \mid i \in \{1,\ldots,k\} \wedge \\ \qquad \widehat{\kappa}_{\text{a name}}(x_i) = \operatorname{send}_{gC} \\ \wedge \widehat{\kappa}_{cnt}(x_1) < \cdots < \widehat{\kappa}_{cnt}(x_k)\}, \omega), & \text{if } k>1. \end{cases} \tag{3}$$

This definition assumes that the very first message of any action sequence of the system is sent by an honest ECU.

5.1.2. Introducing a Phase Class into the Model. In the GenCnt system, only messages with correct counters shall be received and accepted, i.e. their values shall be strictly monotonically increasing. Each counter therefore identifies a phase of the system that starts with sending the message containing it. Hence we use send actions to identify phase classes: Each send actions starts a new phase class, and the phase class ends with the next send action that in turn starts a new phase class. The formal definition of this particular (S,T)-phase class is based on Definition 6:

Definition 9. For $\Upsilon \subseteq \Sigma^*_{g\mathrm{Cnt}}$ *and* $a \in \Sigma_{g\mathrm{Cnt}}$ *with* $\widehat{\kappa}_{a\text{name}}(a) = \operatorname{send}_{gC}$ *we define*

$$\Phi(a,\Upsilon) \coloneqq \Phi\left(\Upsilon, \{a\}, \left\{ a'. \in \sum_{gCnt} \mid \widehat{\kappa}_{\text{aname}}(a') = \operatorname{send}_{gC} \right\}\right). \tag{4}$$

From a recipient's point of view, when having performed a receive action b containing a specific counter, the phase class "activating" this counter starts with the send action that writes this particular message onto the bus. Considering the characteristics of a CAN bus as described in Section 2.1, there cannot be any other send action between these two actions on the bus. Recall that a message manipulated by Eve is considered to have been sent by her. Consequently we do not have two consecutive send_{gC} actions. Hence for each receive action b occuring in a sequence of actions, the corresponding send action, denoted by $\sigma(b)$, is unique. Consequently, each b determines a unique phase class $\Phi(a,\Upsilon)$ with $a=\sigma(b)$. Thus for a specific receive action $b \in \omega$ we can rename the phase class it determines and denote it by $\Phi(\sigma(b),\Upsilon)$. For the sake of completeness, for a send action s we define $\sigma(s) \coloneqq s$.

For the rest of the paper we will use the particular phase class $\Phi(\sigma(b), W_{gCnt})$ determined by a recv_{gC} action b with W_{gCnt} denoting all ECUs' initial knowledge that we assume to be identical. In Section 5.2 we will explain how this phase class can be used to model immediacy and non-repeatability.

5.1.3. Agents' Local View and Initial Knowledge. The definition of the ECUs' local view must take into account that they can see the messages sent on the bus they are connected to but cannot see who sent them nor the local parameters of the sender. Further, except for these send actions, ECUs can only see their own actions. Hence for all $P \in \mathbb{P}_{gCnt}$ and for all $a \in \Sigma_{gCnt}$ we define λ_P as follows:

(1) $\widehat{\kappa}_{\text{ecu}}(a) = \lambda_{P(a)} \coloneqq a$

(2) $\widehat{\kappa}_{\text{ecu}}(a) \neq P \wedge \widehat{\kappa}_{\text{aname}}(a) \in \{\text{read}_{gC}, \text{recv}_{gC}, \text{loseCnt}_{gC}\} \Rightarrow \lambda_P(a) \coloneqq \varepsilon$

(3) $\widehat{\kappa}_{\text{ecu}}(a) \neq P \wedge \widehat{\kappa}_{\text{aname}}(a) = \operatorname{send}_{gC} \Rightarrow \lambda_P(a) \coloneqq (\operatorname{send}_{gC}, \text{bus}, \widehat{\kappa}_{\text{mackey}}(a), \widehat{\kappa}_{\text{msgid}}(a), \widehat{\kappa}_{\text{msg}}(a), \widehat{\kappa}_{cnt}(a))$

Agents' Initial Knowledge The agents' initial knowledge captures the constraints and assumptions that we know to hold for our system. If not specified otherwise, the properties refer to $\omega \in W_{gCnt}$.

Prop. A1. A receive action on bus is always preceded by the corresponding send action that writes the message onto the bus. Obviously, the parameter values of mackey, msgid, msg and cnt in b and $\sigma(b)$ are identical (we forgo the formal specification of this statement). The only actions that can happen in between are read_{gC}, recv_{gC} and loseCnt_{gC} actions by ECUs other than sender and receiver. Formally:

For all $b \in \Sigma_{gCnt}$ with $\widehat{\kappa}_{\text{aname}} = \text{recv}_{gC}$ holds

(1) $\text{prec}_{W_{gCnt}}(\sigma(b), b)$

(2) $\forall \nu \in \Phi(W_{bCnt}, \{\sigma(b)\}, \{b\}) \forall a \in \operatorname{alph}(\nu)$: $\widehat{\kappa}_{\text{ecu}}(a) = \widehat{\kappa}_{\text{ecu}}(\sigma(b)) \vee \widehat{\kappa}_{\text{aname}}(a) = \operatorname{send}_{gC} \Rightarrow a = \text{pre}_1(\nu) = \sigma(b)$ and $\widehat{\kappa}_{\text{ecu}}(a) = \widehat{\kappa}_{\text{ecu}}(b) \Rightarrow a = \text{suf}_1(\nu) = b$

Prop. A2. Only members of $\mathcal{ECU}_{gC}$ own and can use `key`. Since Eve does not own this key and honest ECUs use only `key` to generate and verify a MAC, the MAC key contained in a send or receive action being equal to the ECU's key and this being equal to `key` is equivalent to the ECU being member of $\mathcal{ECU}_{gC}$.

$\forall a \in \operatorname{alph}(\omega) : \widehat{\kappa}_{\text{aname}}(a) \in \{\operatorname{send}_{gC}, \operatorname{recv}_{gC}\} \Rightarrow (\widehat{\kappa}_{\text{mackey}}(a) = \widehat{\kappa}_{\text{ecukey}}(a) = \text{key} \Leftrightarrow \widehat{\kappa}_{\text{ecu}}(a) \in \mathcal{ECU}_{gC})$.

Prop. A3. A recv_{gC} action performed by an honest ECU (i.e. a member of $\mathcal{ECU}_{gC}$) must be preceded by the respective send action of an agent having generated the MAC, i.e. owning the key used for MAC generation:

$\forall \text{ecu} \in \mathcal{ECU}_{gC}$: $\text{prec}_{W_{gCnt}}(\{(\operatorname{send}_{gC}, \text{ecu}', \text{ecukey}', \text{ecucnt}', \text{prevcnt}', \text{bus}, \text{mackey}, \text{msgid}, \text{msg}, \text{cnt}) \mid \text{ecukey}' = \text{mackey} = \text{key}\}, (\operatorname{recv}_{gC}, \text{ecu}, \text{ecukey}, \text{ecucnt}, \text{prevcnt}, \text{bus}, \text{mackey}, \text{msgid}, \text{msg}, \text{cnt}))$

Again, obviously, the parameter values of mackey, msgid, msg and cnt in b and the send action are identical.

Prop. A4. The parameter prevcnt of an action performed by an honest ECU denotes the local message counter the ECU has used in its previous action. For the very first action of an ECU it is defined as the minimal value (which we assume without loss of generality to be equal to 1) of $bCnt$.

$\forall a \in \text{alph}(\omega): \widehat{\kappa}_{\text{ecu}} \in \mathscr{E}CU_{gC} \Rightarrow \widehat{\kappa}_{\text{prevcnt}}(a) = \widehat{\kappa}_{\text{ecucnt}}(\text{prevact}(a, \widehat{\kappa}_{\text{ecu}}(a), \omega))$. If for all $a_i \in \omega$ with $\text{actCnt}(a_i, \omega) < \text{actCnt}(a, \omega)$ holds $\widehat{\kappa}_{\text{ecu}}(a_i) \neq \widehat{\kappa}_{\text{ecu}}(a)$, it follows $\widehat{\kappa}_{\text{prevcnt}}(a) = 1$.

Prop. A5. FvM is the only ECU that sends synchronization messages. It does not perform any other action.

$\forall a \in \text{alph}(\omega): \widehat{\kappa}_{\text{ecu}}(a) = \text{FvM} \Leftrightarrow \widehat{\kappa}_{\text{aname}}(a) = \text{send}_{gC} \wedge \widehat{\kappa}_{\text{msgid}}(a) = \text{sync} \wedge \widehat{\kappa}_{\text{ecucnt}}(a) = \widehat{\kappa}_{\text{msg}}(a)$.

Prop. A6. It is obvious that synchronization messages including a wrong (i.e. too small) counter value open possibilities for all kinds of attacks. Hence we assume that a synchronization message sent by FvM always contains the correct counter value according to Definition 8.

$\forall x \in \text{pre}(\omega): \widehat{\kappa}_{\text{ecu}}(\text{suf}_1(x)) = \text{FvM} \Rightarrow \widehat{\kappa}_{\text{msg}}(\text{suf}_1(x)) = \text{corCnt}_{gC}(\text{suf}_1(x), x)$.

Prop. A7. When an honest agent different to FvM receives (i.e. accepts) a synchronization message, it verifies that the message's payload (which contains the counter) is greater than the local counter used in its previous action and then sets its local counter to the value of the message counter:

$\forall a \in \text{alph}(\omega): \widehat{\kappa}_{\text{aname}}(a) = \text{recv}_{gC} \wedge \widehat{\kappa}_{\text{ecu}}(a) \in \text{ECU}_{gC} \setminus \{\text{FvM}\} \Rightarrow \widehat{\kappa}_{\text{msg}}(a) = \widehat{\kappa}_{\text{ecucnt}}(a) \geq \widehat{\kappa}_{\text{prevcnt}}(a) + 1$.

Prop. A8. An honest agent other than FvM only sends functional messages. When doing so, it increments the counter used in its previous action by 1, uses this value as its new local counter value and as the value of cnt for MAC generation.

$\forall a \in \text{alph}(\omega): \widehat{\kappa}_{\text{aname}}(a) = \text{send}_{gC} \wedge \widehat{\kappa}_{ecu}(a) \in \text{ECU}_{gC} \setminus \{\text{FvM}\} \Leftrightarrow \widehat{\kappa}_{\text{msgid}}(a) = \text{fmsg} \wedge \widehat{\kappa}_{\text{ecucnt}}(a) = \widehat{\kappa}_{\text{cnt}}(a) = \widehat{\kappa}_{\text{prevcnt}}(a) + 1$.

Prop. A9. When an honest agent different to FvM receives (i.e. accepts) a functional message, it verifies that the message's counter is greater than the local counter used in its previous action and then sets its local counter to the value of the message counter:

$\forall a \in \text{alph}(\omega): \widehat{\kappa}_{\text{aname}}(a) = \text{recv}_{gC} \wedge \widehat{\kappa}_{\text{ecu}}(a) \in \text{ECU}_{gC} \setminus \{\text{FvM}\} \Rightarrow \widehat{\kappa}_{\text{cnt}}(a) = \widehat{\kappa}_{\text{ecucnt}}(a) \geq \widehat{\kappa}_{\text{prevcnt}}(a) + 1$.

Prop. A10. An important property of the generic counter system GenCnt is that an ECU increases its counter value only in case it has received and accepted a message with a bigger counter value. Hence an action read_{gC} by an honest ECU does not change ecu's local counter value: $\forall a \in \text{alph}(\omega): \widehat{\kappa}_{\text{aname}}(a) = \text{read}_{gC} \wedge \widehat{\kappa}_{\text{ecu}}(a) \in \text{ECU}_{gC} \Rightarrow \widehat{\kappa}_{\text{ecucnt}}(a) = \widehat{\kappa}_{\text{prevcnt}}(a)$

Prop. A11. An action loseCnt_{gC} performed by an honest ECU resets the ECU's counter value to a value smaller than the correct one:

$\forall x \in \text{pre}(\omega): \widehat{\kappa}_{\text{aname}}(\text{suf}_1(x)) = \text{loseCnt}_{gC} \wedge \widehat{\kappa}_{\text{ecu}}(\text{suf}_1(x)) \in \text{ECU}_{gC} \Rightarrow \widehat{\kappa}_{\text{ecucnt}}(\text{suf}_1(x)) < \text{corCnt}_{gC}(\text{suf}_1(x), x)$

Prop. A12. When an honest ECU performs two recv_{gC} actions with its local genCnt value of the first one being bigger than or equal to the local genCnt value of the second one, it must have performed a loseCnt_{gC} action in between.

For $\omega = x_1 \ldots x_k, 1 \leq i < l < j \leq k$, if $\widehat{\kappa}_{\text{aname}}(x_i) = \widehat{\kappa}_{\text{aname}}(x_j) = \text{recv}_{gC}$ and $\widehat{\kappa}_{\text{ecu}}(x_i) = \widehat{\kappa}_{\text{ecu}}(x_j) \in \text{ECU}_{gC}$ and $\widehat{\kappa}_{\text{ecucnt}}(x_i) \geq \widehat{\kappa}_{\text{ecucnt}}(x_j)$ then there exists $x_l \in \text{alph}(\omega)$ with $\widehat{\kappa}_{\text{aname}}(x_l) = \text{loseCnt}_{gC}$ and $\widehat{\kappa}_{\text{ecu}}(x_l) = \widehat{\kappa}_{\text{ecu}}(x_i) = \widehat{\kappa}_{\text{ecu}}(x_j)$.

This concludes our system model specification. In the next section, we will show that the model allows certain states which violate a property that can be used for the specification of authenticity, immediacy and non-repeatability.

5.2. Formal Verification of the Generic Counter Concept. As stated in Section 2.5, the security requirements the generic counter system (denoted by B_{gCnt}) shall satisfy are data origin authenticity, immediacy and non-repeatability. More precisely, an honest ECU shall accept only messages authentically generated and sent by another honest ECU, thus providing data origin authenticity. Further, the message must contain the correct counter which ensures that no counter is accepted twice (since the correct counter is strictly monotonically increasing), thus providing non-repeatability. In order to express this, we use the phase class $\Phi(\sigma(b), W_{gCnt})$ as defined in Definition 9 with b being a recv_{gC} action. Each time an honest ecu receives a message, the message must authentically for ecu have been sent by a member of the same group, and this send action must be the one to trigger ecu's recv_{gC} action b, i.e. must be the start action $\sigma(b)$ of the phase class determined by b. Since the time period between sending and receiving messages on a CAN bus is very short, we can assume that it never exceeds the specified limit which implies immediacy. This can be formalized as follows:

Theorem 1. *Let* $\omega \in B_{gCnt}$ *and* $b \coloneqq (\text{recv}_{gC}, ecu, ecukey, ecucnt, prevcnt, bus, mackey, msgid, msg, cnt) \in alph(\omega)$ *with* $ecu \in \mathscr{E}CU_{gC}$. *Then the following property holds:*

$$\text{authWiPhase}\Big(\{(\text{send}_{gC}, \text{ecu}', \text{ecukey}', \text{ecucnt}', \text{prevcnt}', \text{bus}, \text{mackey}, \text{msgid}, \text{msg}, \text{cnt}) \mid \text{ecu}' \in \text{ECU}_{gC}\}, b, \text{ecu}, \Phi(\sigma(b), W_{gCnt})\Big) \tag{5}$$

Proof 1. Assume one of the honest ECUs different to FvM that is member of the group receives (i.e. accepts) a message. Without loss of generality assume it is $\text{ECU}_1^{gC} \in \mathscr{E}CU_{gC}$ and $b \coloneqq (\text{recv}_{gC}, \text{ECU}_1^{gC}, \text{ecukey}, \text{ecucnt}, \text{prevcnt}, \text{bus}, \text{mackey}, \text{msgid}, \text{msg}, \text{cnt}) \in \text{alph}(\omega)$ for some $\omega \in B_{gCnt}$. By definition, $\lambda_{\text{ECU}_1^{gC}}$ keeps this action, thus b is also contained in each $x \in \lambda^{-1}_{\text{ECU}_1^{gC}}(\lambda_{\text{ECU}_1^{gC}}(\omega))$. Further, b is contained in $\omega \in B_{gCnt} \subseteq W_{gCnt}$. So let $x \in \lambda^{-1}_{\text{ECU}_1^{gC}}(\lambda_{\text{ECU}_1^{gC}}(\omega)) \cap W_{gCnt}$ arbitrarily chosen. Since $\text{ECU}_1^{gC} \in \mathscr{E}CU_{gC}$, Prop.A2 implies that ecukey = mackey = key. Further, by Prop.A3, there is an action $a_1 \coloneqq (\text{send}_{gC}, \text{ecu}_1, \text{ecukey}_1, \text{ecucnt}_1, \text{prevcnt}_1, \text{bus}, \text{mackey}, \text{msgid}, \text{msg}, \text{cnt}) \in \text{alph}(x)$ before ECU_1^{gC}'s receive

action containing the same message, message ID and counter value and with $\text{ecukey}_1 = \text{mackey} = \text{key}$. Applying again Prop.A2 it follows $\text{ecu}_1 \in \text{ECU}_{gC}$. Hence the message received in b has authentically for ECU_1^{gC} been sent by a member of ECU_{gC}, i.e. data origin authenticity is satisfied.

Therefore $\text{ecu}_1 = \text{FvM}$ and $\text{msgid} = \text{sync}$ (Prop.A5) or $\text{ecu}_1 = \text{ECU}_2^{gC}$ and $\text{msgid} = \text{fmsg}$ (Prop.A8) (we disregard the fact that in principle ECU_1^{gC} could itself be the originator of this message and assume this issue to be addressed by e.g. unique message IDs). By Prop.A1 b is preceded by $\sigma(b) = (\text{send}_{gC}, \text{ecu}', \text{ecukey}', \text{ecucnt}', \text{bus}, \text{mackey}, \text{msgid}, \text{msg}, \text{cnt})$ that starts the phase class identified by b. By definition, the local view of ECU_1^{gC} does not reveal the sender, hence assume $\text{ecu}' \neq \text{ecu}_1$ and $\sigma(b) \neq a_1$, i.e. assume that the authentic action a_1 is not performed in the required phase class. Assume further that after having performed their respective last actions before a_1 (denoted by a_2 and a_3, respectively), ecu_1 and ECU_1^{gC} are synchronized, i.e. own the same counter which is the correct one for these actions. Let us assume ECU_1^{gC} performs a_3, ecu_1 performs a_2 and $\widehat{\kappa}_{\text{ecucnt}}(a_2) = \widehat{\kappa}_{\text{ecucnt}}(a_3) = k = \text{corCnt}_{gC}(a_2, x) = \text{corCnt}_{gC}(a_3, x)$.

Assume $\text{ecu}_1 = \text{ECU}_2^{gC}$ and $\widehat{\kappa}_{\text{msgid}}(a_1) = \texttt{fmsg}$. Then Prop.A8 implies $\widehat{\kappa}_{\text{ecucnt}}(a_1) = \widehat{\kappa}_{\text{prevcnt}}(a_1) + 1 = \widehat{\kappa}_{\text{cnt}}(a_1)$. Since a_2 is the last action of ecu_1 before a_1, Prop.A4 implies that $\widehat{\kappa}_{\text{prevcnt}}(a_1) = \widehat{\kappa}_{\text{ecucnt}}(a_2)$. Prop.A3 implies $\text{cnt} = \widehat{\kappa}_{\text{cnt}}(b) = \widehat{\kappa}_{\text{cnt}}(a_1)$ and it follows $\widehat{\kappa}_{\text{cnt}}(a_1) = \widehat{\kappa}_{\text{ecucnt}}(a_1) = \widehat{\kappa}_{\text{ecucnt}}(a_2) + 1 = k + 1$. This situation, depicted in Table 2 , is the basis for the subsequent case-by-case analysis (note that it is irrelevant whether a_3 precedes a_2 or vice versa). □

5.2.1. Losing the counter. Assume that ECU_1^{gC} receives the message sent by ecu_1 in a_1 by performing an action a_4 (i.e. $\sigma(a_4) = a_1$). Then $\widehat{\kappa}_{\text{cnt}}(a_4) = \widehat{\kappa}_{\text{cnt}}(a_1) = k + 1$ (Prop.A1) and Prop.A9 implies $\widehat{\kappa}_{\text{ecucnt}}(a_4) = \widehat{\kappa}_{\text{cnt}}(a_4) = k + 1$. Prop.A9 also requires $\widehat{\kappa}_{\text{ecucnt}}(a_4) \geq \widehat{\kappa}_{\text{prevcnt}}(a_4) + 1$. This is the case, as by Prop.A4 we can conclude $\widehat{\kappa}_{\text{prevcnt}}(a_4) = \widehat{\kappa}_{\text{ecucnt}}(a_3)$ and thus $k + 1 = \widehat{\kappa}_{\text{ecucnt}}(a_4) \geq \widehat{\kappa}_{\text{ecucnt}}(a_3) + 1$ which by the assumption of ECU_1^{gC} and ecu_1 being synchronized before a_1 is equal to $\widehat{\kappa}_{\text{ecucnt}}(a_2) + 1 = k + 1$, hence $\widehat{\kappa}_{\text{ecucnt}}(a_4) = k + 1 \geq k + 1$ is satisfied. Since with action b, ECU_1 receives and accepts $\text{cnt} = k + 1$, Prop.A12 implies that ECU_1^{gC} performs an action $a_5 := (\text{loseCnt}_{gC}, \text{ECU}_1^{gC}, \text{ecucnt}_5, \text{prevcnt}_5, \text{bus})$ between a_4 and b, and Prop.A11 implies that ecucnt_5 is smaller than the correct counter value for this action which in turn is equal to or bigger than $\widehat{\kappa}_{\text{cnt}}(a_5) = k + 1$. Assume that between a_4 and a_5 there have been k' send actions by members of $\mathcal{ECU}_{gC}$ other than ECU_1^{gC} with correct counters, increasing its value to $k + 1 + k'$ without changing ECU_1^{gC}'s counter value (e.g. because it does not perform any action other than a_5 between a_4 and b). It follows $\text{ecucnt}_5 < \text{corCnt}_{gC}(a_5, x) = k + 1 + k'$. On the other hand, in b ECU_1^{gC} receives and accepts the message sent in $\sigma(b)$ with the counter $\text{cnt} = k + 1$. So assuming ECU_1^{gC}'s loseCnt_{gC} action a_5 to be its last action before b, Prop.A4 and Prop.A9 imply $k + 1 = \widehat{\kappa}_{\text{cnt}}(b) = \widehat{\kappa}_{\text{ecucnt}}(b) \geq \widehat{\kappa}_{\text{prevcnt}}(b) + 1 = \text{ecucnt}_5$. Both inequalities are satisfied for $\text{ecucnt}_5 \leq k + 1 - 1 = k$. Thus ECU_1^{gC} may very well receive and accept the message sent in $\sigma(b)$ by $\text{ecu}\prime \notin \mathcal{ECU}_{gC}$.

The resulting sequence of actions is depicted in Table 3. While it satisfies data origin authenticity, it violates immediacy, assuming that only the time period between writing a message onto the bus and reading it does not exceed the specified limit. It also violates non-repeatability as the message sent in a_1 is accepted twice.

It is not surprising that counter loss without timely synchronization opens up attack possibilities. The same result can be shown in case $\text{ecu}_1 = \text{FvM}$ sends a synchronization message in a_1. We then need to consider the fact that between a_2 and a_1, ECU_2^{gC} may have sent n messages that increase the correct counter of a_1 accordingly. Further, instead of applying Prop.A8 we need to take into account that the counter is sent as the message's payload, i.e. modeled by the parameter msg, and apply Prop.A5.

We will now investigate whether sending of synchronization messages prohibits the above described attack. Assume therefore that with the loseCnt_{gC} action ECU_1^{gC} sets its local genCnt value to $k'\prime < k + 1 + k'$ and that between the loseCnt_{gC} action and b, ECU_1^{gC} receives one or more synchronization messages with a_6 being the last one before b. Since $\text{cnt} = k + 1$ is the counter accepted by ECU_1^{gC} in b and since the counter sent in a synchronization message is contained in its payload, $\widehat{\kappa}_{\text{msg}}(a_6) = \text{msg}_6 \in [k'' + 1, k]$. Assume further these are the only messages sent on the bus. Then again by Prop.A4 and Prop.A7, $\text{cnt} = k + 1 = \widehat{\kappa}_{\text{cnt}}(b) = \widehat{\kappa}_{\text{ecucnt}}(b) \geq \widehat{\kappa}_{\text{prevcnt}}(b) + 1 = \widehat{\kappa}_{\text{ecucnt}}(a_6) + 1 = \widehat{\kappa}_{\text{msg}}(a_6) + 1 = \text{msg}_6 + 1 \geq k'' + 2$ which implies $k'' \leq \text{msg}_6 - 1 \leq k + 1 - 2 = k - 1$. According to Prop.A1, a_6 is preceded by an action $\sigma(a_6)$ in which the synchronization message received by ECU_1^{gC} is written to the bus. The property further implies that between $\sigma(a_6)$ and a_6, ECU_1^{gC} does not perform any further action, hence $\sigma(a_6)$ happens after ECU_1^{gC}'s loseCnt_{gC} action and in particular after a_1. Recall now that we have assumed k to be the correct counter value of both a_2 and a_3 and that $\widehat{\kappa}_{\text{cnt}}(a_1) = k + 1$. If $\widehat{\kappa}_{\text{ecu}}(\sigma(a_6))$ was FvM, Prop.A6 would imply $\widehat{\kappa}_{\text{msg}}(\sigma(a_6)) \geq k + 1 + 1 = k + 2$. Yet what ECU_1^{gC} accepts in a_6 is $\text{msg}_6 \leq k + 1 - 1 = k$. Thus $\widehat{\kappa}_{\text{ecu}}(\sigma(a_6)) \neq \text{FvM}$ and since by Prop.A8 an honest agent other than FvM only sends functional messages, it follows $\widehat{\kappa}_{\text{ecu}}(\sigma(a_6)) = \text{Eve}$.

While by Prop.A3, a_6 is preceded by an action $a_7 := (\text{send}_{gC}, \text{FvM}, \ldots, \text{sync}, \ldots \text{msg}_6, \ldots)$, this does not necessarily interfere with the attack we are constructing here, assuming that ECU_1^{gC} does not receive (i.e. accept) this message but only performs a read_{gC} action which does not change its local counter. This attack is illustrated in Table 4.

This attack uses an important characteristic of the generic counter system GenCnt, captured in Prop.A10: It causes ECU_1^{gC} to keep the too small and thus incorrect counter since it does not correctly receive and accept the synchronization messages sent by FvM between a_5 and b. The attack again violates immediacy and non-repeatability.

5.2.2. Not losing the counter. Let us now consider the case in which ECU_1^{gC} only performs read_{gC} actions between a_3 and b and in particular does not perform an action a_4, i.e. does not receive and accept the message sent by ecu_1 in a_1. As above, by Prop.A1 and Prop.A3 we know $\widehat{\kappa}_{\text{cnt}}(a_1) = \widehat{\kappa}_{\text{cnt}}(b) = \widehat{\kappa}_{\text{cnt}}$

Table 2: Developing possible sequences

a_3	last action by ECU_1^{gC} before a_1	$\widehat{\kappa}_{\mathrm{ecucnt}}(a_2) = k$ (k is correct counter)
a_2	last action by ecu_1 before a_1	$\widehat{\kappa}_{\mathrm{ecucnt}}(a_3) = k$
⋮		
	no actions by ECU_1^{gC} and ecu_1	
⋮		
a_1	$(\mathrm{send}_{gC}, \mathrm{ecu}_1, \mathrm{ecukey}, \mathrm{ecucnt}_1, \mathrm{prevcnt}_1, \mathrm{bus}, \mathrm{mackey}, \mathrm{msgid}, \mathrm{msg}, \mathrm{cnt})$	$\mathrm{ecucnt}_1 = \mathrm{prevcnt}_1 + 1 = \widehat{\kappa}_{\mathrm{ecucnt}}(a_2) + 1 = k + 1 = \mathrm{cnt}$
⋮		
$\sigma(b)$	$(\mathrm{send}_{gC}, \mathrm{ecu}', \mathrm{ecukey}', \mathrm{ecucnt}', \mathrm{prevcnt}', \mathrm{bus}, \mathrm{mackey}, \mathrm{msgid}, \mathrm{msg}, \mathrm{cnt})$	$\mathrm{cnt} = k + 1$ by Prop.A1
b	$(\mathrm{recv}_{gC}, \mathrm{ECU}_1^{gC}, \mathrm{ecukey}, \mathrm{ecucnt}, \mathrm{prevcnt}, \mathrm{bus}, \mathrm{mackey}, \mathrm{msgid}, \mathrm{msg}, \mathrm{cnt})$	$\mathrm{ecucnt} = \mathrm{cnt} = k + 1$

Table 3: A first attack sequence.

a_3	action by ECU_1^{gC}	$\widehat{\kappa}_{\mathrm{ecucnt}}(a_2) = k$
a_2	action by ecu_1	$\widehat{\kappa}_{\mathrm{ecucnt}}(a_3) = k$
⋮		
	no actions by ECU_1^{gC} and ecu_1	
⋮		
a_1	$(\mathrm{send}_{gC}, \mathrm{ecu}_1, \mathrm{ecukey}, \mathrm{ecucnt}_1, \mathrm{prevcnt}_1, \mathrm{bus}, \mathrm{mackey}, \mathrm{msgid}, \mathrm{msg}, \mathrm{cnt})$	$\mathrm{ecucnt}_4 = \mathrm{cnt} = k + 1$ $\widehat{\kappa}_{\mathrm{ecucnt}}(a_2) + 1 = k + 1 = \mathrm{cnt}$
a_4	$(\mathrm{recv}_{gC}, \mathrm{ECU}_1^{gC}, \mathrm{ecukey}, \mathrm{ecucnt}_4, \mathrm{prevcnt}_4, \mathrm{bus}, \mathrm{mackey}, \mathrm{msgid}, \mathrm{msg}, \mathrm{cnt})$	$\mathrm{ecucnt}_4 = \mathrm{cnt} = k + 1$
⋮		
a_5	$(\mathrm{loseCnt}_{gC}, \mathrm{ECU}_1^{gC}, \mathrm{ecucnt}_5, \mathrm{prevcnt}_5, \mathrm{bus})$	$\mathrm{ecucnt}_5 < k + 1 + k'$
⋮		
	no action by ECU_1^{gC}	
⋮		
$\sigma(b)$	$(\mathrm{send}_{gC}, \mathrm{ecu}', \mathrm{ecukey}', \mathrm{ecucnt}', \mathrm{prevcnt}', \mathrm{bus}, \mathrm{mackey}, \mathrm{msgid}, \mathrm{msg}, \mathrm{cnt})$	$\mathrm{cnt} = k + 1$
b	$(\mathrm{recv}_{gC}, \mathrm{ECU}_1^{gC}, \mathrm{ecukey}, \mathrm{ecucnt}, \mathrm{prevcnt}, \mathrm{bus}, \mathrm{mackey}, \mathrm{msgid}, \mathrm{msg}, \mathrm{cnt})$	$\mathrm{ecucnt} = \mathrm{cnt} = k + 1$

Table 4: The second possible attack sequence.

a_3	action by ECU_1^{gC}	$\widehat{\kappa}_{\mathrm{ecucnt}}(a_2) = k$
a_2	action by ecu_1	$\widehat{\kappa}_{\mathrm{ecucnt}}(a_3) = k$
⋮		
	no actions by ECU_1^{gC} and ecu_1	
⋮		
a_1	$(\mathrm{send}_{gC}, \mathrm{ecu}_1, \mathrm{ecukey}, \mathrm{ecucnt}_1, \mathrm{prevcnt}_1, \mathrm{bus}, \mathrm{mackey}, \mathrm{msgid}, \mathrm{msg}, \mathrm{cnt})$	$\mathrm{ecucnt}_1 = \mathrm{prevent}_1 + 1 =$ $\widehat{\kappa}_{\mathrm{ecucnt}}(a_2) + 1 = k + 1 = \mathrm{cnt}$
a_4	$(\mathrm{recv}_{gC}, \mathrm{ECU}_1^{gC}, \mathrm{ecukey}, \mathrm{ecucnt}_4, \mathrm{prevcnt}_4, \mathrm{bus}, \mathrm{mackey}, \mathrm{msgid}, \mathrm{msg}, \mathrm{cnt})$	$\mathrm{ecucnt}_4 = \mathrm{cnt} = k + 1$
⋮		
a_5	$(\mathrm{loseCnt}_{gC}, \mathrm{ECU}_1^{gC}, \mathrm{ecucnt}_5, \mathrm{prevcnt}_5, \mathrm{bus})$	$\mathrm{ecucnt}_5 < k + 1 + k'$
⋮		
a_7	$(\mathrm{send}_{gC}, \mathrm{FvM}, \ldots, \mathrm{sync}, \ldots, \mathrm{msg}_6, \ldots,)$	
a_8	$(\mathrm{read}_{gC}, \mathrm{ECU}_1^{gC}, \ldots, \mathrm{ecucnt}_8, \mathrm{prevcnt}_8, \mathrm{sync}, \ldots, \mathrm{msg}_6, \ldots)$	$\mathrm{ecucnt}_8 = \mathrm{prevcnt}_8 = \mathrm{ecucnt}_5$
⋮		
$\sigma(a_6)$	$(\mathrm{send}_{gC}, \mathrm{Eve}, \ldots, \mathrm{sync}, \ldots, \mathrm{msg}_6, \ldots,)$	$\mathrm{msg}_6 \in [k'' + 1, k + 1 - 1]$
a_6	$(\mathrm{recv}_{gC}, \mathrm{ECU}_1^{gC}, \mathrm{ecukey}, \mathrm{ecucnt}_6, \mathrm{prevcnt}_6, \ldots, \mathrm{sync}, \ldots, \mathrm{msg}_6, \ldots,)$	$\mathrm{ecucnt}_6 = \mathrm{msg}_6 \leq k + 1 - 1$ $\mathrm{prevent}_6 = \mathrm{ecucnt}_8 = k''$ $\wedge \mathrm{msg}_6 \geq k'' + 1$
$\sigma(b)$	$(\mathrm{send}_{gC}, \mathrm{ecu}', \mathrm{ecukey}', \mathrm{ecucnt}', \mathrm{prevcnt}', \mathrm{bus}, \mathrm{mackey}, \mathrm{msgid}, \mathrm{msg}, \mathrm{cnt})$	$\mathrm{cnt} = k + 1$
b	$(\mathrm{recv}_{gC}, \mathrm{ECU}_1^{gC}, \mathrm{ecukey}, \mathrm{ecucnt}, \mathrm{prevcnt}, \mathrm{bus}, \mathrm{mackey}, \mathrm{msgid}, \mathrm{msg}, \mathrm{cnt})$	$\mathrm{ecucnt} = \mathrm{cnt} = k + 1$

($\sigma(b)$). Since by Prop.A10 a read_{gC} action does not change ECU_1^{gC}'s counter, consecutive application of Prop.A4 to the sequence of these read_{gC} actions implies $\widehat{\kappa}_{\mathrm{prevcnt}}(b) = \widehat{\kappa}_{\mathrm{ecucnt}}(a_3)$ and by Prop.A9 it follows $k + 1 = \mathrm{cnt} = \widehat{\kappa}_{\mathrm{cnt}}(b) = \widehat{\kappa}_{\mathrm{ecucnt}}(b) \geq \widehat{\kappa}_{\mathrm{prevcnt}}(b) + 1 = \widehat{\kappa}_{\mathrm{ecucnt}}(a_3) + 1 = k + 1$. This equation is always satisfied which means that there is no contradiction to $\widehat{\kappa}_{\mathrm{ecu}}(\sigma(b)) = \mathrm{ecu}' \neq \mathrm{ecu}_1$. Therefore this sequence (illustrated in Table 5) is another example for violation of immediacy based on invalidation and replay of messages. Note that it does not violate non-repeatability as the message is only received and accepted once.

The only case in which the desired properties hold is if by performing a_4, ECU_1^{gC} receives and accepts the message sent in a_1 and does not lose the correct counter value, i.e. does not perform a $\mathrm{loseCnt}_{gC}$ action between a_4 and b. In this case ECU_1^{gC} sets its local counter $\widehat{\kappa}_{\mathrm{ecucnt}}(a_4)$ to $k + 1$ (Prop.A9) and without losing the correct counter will not accept the same counter in action b anymore, as $\widehat{\kappa}_{\mathrm{prevcnt}}(b) = \widehat{\kappa}_{\mathrm{ecucnt}}$

Table 5: The third possible attack sequence.

a_3	action by ECU_1^{gC}	$\widehat{\kappa}_{\mathrm{ecucnt}}(a_2) = k$
a_2	action by ecu_1	$\widehat{\kappa}_{\mathrm{ecucnt}}(a_3) = k$
⋮		
	only read_{gC} actions by ECU_1^{gC}	
⋮		
a_1	$(\mathrm{send}_{gC}, \mathrm{ecu}_1, \mathrm{ecukey}, \mathrm{ecucnt}_1, \mathrm{prevcnt}_1, \mathrm{bus}, \mathrm{mackey}, \mathrm{msgid}, \mathrm{msg}, \mathrm{cnt})$	$\mathrm{ecucnt}_1 = \mathrm{prevcnt}_1 + 1 = \widehat{\kappa}_{\mathrm{ecucnt}}(a_2) + 1 = k + 1 = \mathrm{cnt}$
⋮		
	only read_{gC} actions by ECU_1^{gC}	
⋮		
$\sigma(b)$	$(\mathrm{send}_{gC}, \mathrm{ecu}', \mathrm{ecukey}', \mathrm{ecucnt}', \mathrm{prevcnt}', \mathrm{bus}, \mathrm{mackey}, \mathrm{msgid}, \mathrm{msg}, \mathrm{cnt})$	$\mathrm{cnt} = k + 1$
b	$(\mathrm{recv}_{gC}, \mathrm{ECU}_1^{gC}, \mathrm{ecukey}, \mathrm{ecucnt}, \mathrm{prevcnt}, \mathrm{bus}, \mathrm{mackey}, \mathrm{msgid}, \mathrm{msg}, \mathrm{cnt})$	$\mathrm{ecucnt} = \mathrm{cnt} = \mathrm{prevcnt} + 1 = \widehat{\kappa}_{\mathrm{ecucnt}}(a_2) = k + 1$

$(a_4) = k + 1$ and thus $\widehat{\kappa}_{\mathrm{cnt}}(b) = k + 1 \neq \widehat{\kappa}_{\mathrm{prevcnt}}(b) = k + 1$ as is required by Prop.A9.

Our proof indicating possible attacks is based on the fact that certain messages are not received but only read by ECU_1^{gC}. This can easily be accomplished by an attacker with the abilities described in Section 2.2. All she has to do is to monitor the respective message and then invalidate it and all following ones related to the relevant counter by changing a CRC bit or overwriting the message with an error frame. This will cause all ECUs connected to the bus to reject the messages. Since read_{gC} actions do not change the ECUs' local counter, any message containing a bigger counter will still be considered correct.

In Section 7.1 we will discuss the results achieved by the proof and compare them to the formal proofs of the bus counter-based system to be introduced in the next section.

6. Formalization and Verification of BusCount

In this section, we introduce the formal model and verification of our hardware-based counter approach.

6.1. The Formal Bus Counter Model. We model the bus counter-based system (denoted by BusCnt henceforth) as similar as possible to the generic counter model. One important difference is that it does not need a central freshness value master since all ECUs send synchronization messages simultaneously. Hence the set of agents is defined as $\mathbb{P}_{bCnt} = \mathscr{E}CU_{bC} \cup \{\mathrm{Eve}\}$ with $\mathscr{E}CU_{bC} \coloneqq \{\mathrm{ECU}_1^{bC}, \mathrm{ECU}_2^{bC}, \mathrm{ECU}_3^{bC}\}$.

As in GenCnt, the BusCnt system has only one bus bus = bus all agents are being connected to. Further, members of $\mathscr{E}CU_{bC}$ are honest and own the key key, while Eve, not being member of this group, does not own this key. We use the same set of action parameters, but a different specification of agents' behavior (after all, we model a different system). The set of actions Σ_{bCnt} is defined as follows:

(i) $(\mathrm{send}_{bC}, \mathrm{ecu}, \mathrm{ecukey}, \mathrm{ecucnt}, \mathrm{prevcnt}, \mathrm{bus}, \mathrm{mackey}, \mathrm{msgid}, \mathrm{msg}, \mathrm{cnt})$ denotes a send action as described in Section 5.1.1, except that the counter value cnt is covered by the MAC but not transmitted. As in the GenCnt model, the message may be altered (by a technical error or by Eve) after having been sent and may thus only cause a read_{bC} action (see below).

(ii) $(\mathrm{read}_{bC}, \mathrm{ecu}, \mathrm{ecukey}, \mathrm{ecucnt}, \mathrm{prevcnt}, \mathrm{bust}, \mathrm{mackey}, \mathrm{msgid}, \mathrm{msg}, \mathrm{cnt})$ denotes agent $\mathrm{ecu} \in \mathbb{P}_{bCnt}$ reading a message without processing it afterwards. In contrast to the respective read_{gC} action of the GenCnt model, the read_{bC} action of the BusCnt model changes the state of an ecu being member of $\mathscr{E}CU_{bC}$ by decrementing its local bCnt value (stored with the last action and thus modeled by the parameter prevcnt) (see Prop.B13 in Section 6.1.1 below). This captures the fact that ecu reacts to the "start of message" bit on bus but discards the respective message (e.g. because the CRC verification fails) in which case it does not perform the receive action. Note that the sender of a message always reads its own action bey performing a read_{bC} action.

(iii) $(\mathrm{recv}_{bC}, \mathrm{ecu}, \mathrm{ecukey}, \mathrm{ecucnt}, \mathrm{prevcnt}, \mathrm{bus}, \mathrm{mackey}, \mathrm{msgid}, \mathrm{msg}, \mathrm{cnt})$ denotes the successful reception and processing of a message by $\mathrm{ecu} \in \mathbb{P}_{bCnt}$.

(iv) As in the GenCnt system, with $(\mathrm{loseCnt}_{bC}, \mathrm{ecu}, \mathrm{ecucnt}, \mathrm{prevcnt}, \mathrm{bus})$ we model the fact that $\mathrm{ecu} \in \mathbb{P}_{bCnt}$ has lost the correct counter value for some reason and thus is no longer synchronized. Since in the BusCnt system counter values decrease, its counter is set to a value bigger than the correct counter value (see Section 6.1.1 for more details).

6.1.1. Agents' Local View and Initial Knowledge. Again, the agents' local view is defined analogously to the generic counter model: All agents see their own actions completely and see the messages sent on the CAN bus they are connected to but cannot see who sent them nor the values of parameters stored locally by the sender. Further, agents cannot see actions read_{bC}, recv_{bC} and $\mathrm{loseCnt}_{bC}$ performed by other agents.

As already pointed out in Section 5.1.3, with specifying the agents' initial knowledge we capture the characteristics of our system. In the following, we list all properties we assume to be satisfied by the agents' initial knowledge W_{bCnt} with reference to the respective property in Section 5.1.3 (if any) in which case we omit the formalization. Analogously to Section 5.1.3, if not specified otherwise, the properties refer to $\omega \in W_{bCnt}$. We denote the correct counter for a specific action a in ω by $\mathrm{corCnt}_{bC}(a, \omega)$. In Lemma 2 (see Section 6.2) we will show how its value is determined.

Prop. B1 (analogous to first statement of Prop.A1). A read_{bC} and recv_{bC} action, respectively, on bus is always preceded by the corresponding send action that writes the message onto the bus. Obviously, the parameter values of mackey, msgid, msg and cnt in b and $\sigma(b)$ are identical (we forgo the formalization of the latter statement).

For all $b \in \Sigma_{bCnt}$ with $\widehat{\kappa}_{\text{aname}} \in \{\text{read}_{bC}, \text{recv}_{bC}\}$ holds $\text{prec}_{W_{bCnt}}(\sigma(b), b)$.

Prop. B2 (Prop.A2). Only members of $\mathcal{ECU}_{bC}$ own and can use key = key. Since Eve does not own this key and honest ECUs use only `key` to generate and verify a MAC, the MAC key contained in a send or receive action being equal to key = `key` is equivalent to the ECU being member of $\mathcal{ECU}_{bC}$.

Prop. B3 (Prop.A3). A recv_{bC} action performed by an honest ECU (i.e. a member of $\mathcal{ECU}_{bC}$) must be preceded by the respective send action of an agent having generated the MAC, i.e. owning the key used for MAC generation. Again, obviously, the parameter values of mackey, msgid, msg and cnt in b and $\sigma(b)$ are identical.

Prop. B4 (Prop.A4). The parameter prevcnt of an action performed by an honest ECU denotes the local bCnt value as result of the ECU's previous action. For the very first action of an ECU it is defined as the maximal value of *bCnt*, denoted by $bCnt_{\max}$. Formally:

$\forall a \in \text{alph}(\omega)$: $\widehat{\kappa}_{\text{prevcnt}}(a) = \widehat{\kappa}_{\text{ecucnt}}(\text{prevact}(a, \widehat{\kappa}_{\text{ecu}}(a), \omega))$. If for all $a_i \in \omega$ with $\text{actCnt}(a_i, \omega) < \text{actCnt}(a, \omega)$ holds $\widehat{\kappa}_{\text{ecu}}(a_i) \neq \widehat{\kappa}_{\text{ecu}}(a)$, it follows $\widehat{\kappa}_{\text{prevcnt}}(a) = bCnt_{\max}$.

Prop. B5. In Section 7.2.2 we will discuss which starting value of *bCnt* to choose in order to avoid counter overflow. Further, as explained in Section 2.2, we assume that memory failures and attacks cannot cause counter overflow. Hence we can assume that such a failure never results into a local counter value stored by an ECU being smaller than the correct one (see Prop.B17 below). For our formal model we assume that the local counter value of ECUs is always sufficiently large such that counter decrementation can result into the value 0 only in the last phase class of an action sequence. Formally:

$\forall a \in \text{alph}(\omega)$: $\widehat{\kappa}_{\text{prevcnt}}(a) > 0$

Prop. B6 (analog to Prop.A8). When an honest ECU sends a synchronization message, it includes as its message payload the local bCnt value of its previous action decremented by 1 but does not change the local bCnt value.

$\forall a \in \text{alph}(\omega)$: $\widehat{\kappa}_{\text{aname}}(a) = \text{send}_{bC} \wedge \widehat{\kappa}_{\text{ecu}}(a) \in \mathcal{ECU}_{bC} \wedge \widehat{\kappa}_{\text{msgid}}(a) = \text{sync} \Rightarrow \widehat{\kappa}_{\text{msg}}(a) = \widehat{\kappa}_{\text{prevcnt}}(a) - 1 \wedge \widehat{\kappa}_{\text{ecucnt}}(a) = \widehat{\kappa}_{\text{prevcnt}}(a)$.

Prop. B7. We assume that there always exists an ECU owning the correct counter value. Since our synchronization concept utilizes the mechanism used for collision resolving (a 0 written to the CAN bus always overwrites a 1), the correct counter always overwrites any incorrect one. Therefore a send_{bC} action containing a synchronization message that is actually performed by an honest ECU always contains the correct counter. Formally:

$a \in \text{alph}(\omega) \wedge \widehat{\kappa}_{\text{aname}}(a) = \text{send}_{bC} \wedge \widehat{\kappa}_{\text{ecu}}(a) \in \mathcal{ECU}_{bC} \wedge \widehat{\kappa}_{\text{msgid}}(a) = \text{sync} \Rightarrow \widehat{\kappa}_{\text{msg}}(a) = \text{corCnt}_{bC}(a, \omega)$.

Prop. B8. When monitoring a synchronization message being written to the bus, an honest ECU decrements its previously used counter value by 1 and verifies that the result is less or equal to the counter sent as the message's payload. The error frame parameter being equal to *no* indicates that this check has been successful (and that the MAC check that we do not formalize explicitly has been successful as well). It then uses this value as its new local counter.

$\forall a \in \text{alph}(\omega)$: $\widehat{\kappa}_{\text{aname}}(a) \in \{\text{read}_{bC}, \text{recv}_{bC}\} \wedge \widehat{\kappa}_{\text{ecu}}(a) \in \mathcal{ECU}_{bC} \wedge \widehat{\kappa}_{\text{msgid}}(a) = \text{sync} \wedge \widehat{\kappa}_{\text{errorFrame}}(a) = \text{no} \Rightarrow \widehat{\kappa}_{\text{ecucnt}}(a) = \widehat{\kappa}_{\text{msg}}(a) \leq \widehat{\kappa}_{\text{prevcnt}}(a) - 1$.

While a is actually a recv_{bC} action, our proofs do not depend on distinguishing between read_{bC} and recv_{bC} actions of synchronization messages.

Prop. B9 (analog to Prop.A8). When an honest ECU sends a functional message, it includes as its counter value the local bCnt value of its previous action decremented by 1 but does not change the local bCnt value. (It changes the value of ecucnt with the action of reading its own message that the ECU performs simultaneously to sending, see Prop.B10 and Prop.B13.)

$\forall a \in \text{alph}(\omega)$: $\widehat{\kappa}_{\text{aname}}(a) = \textit{send}_{bC} \wedge \widehat{\kappa}_{\text{ecu}}(a) \in \mathcal{ECU}_{bC} \wedge \widehat{\kappa}_{\text{msgid}}(a) = \text{fmsg} \Rightarrow \widehat{\kappa}_{\text{cnt}}(a) = \widehat{\kappa}_{\text{prevcnt}}(a) - 1 \wedge \widehat{\kappa}_{\text{ecucnt}}(a) = \widehat{\kappa}_{\text{prevcnt}}(a)$.

Prop. B10 (analog to Prop.A9). When an honest ECU receives and accepts a functional message, it decrements its previously used counter value by 1 and verifies that the message's cnt value is equal to the result. It then sets its local bCnt value ecucnt to the message's counter.

$\forall a \in \text{alph}(\omega)$: $\widehat{\kappa}_{\text{aname}}(a) = \text{recv}_{bC} \wedge \widehat{\kappa}_{\text{ecu}}(a) \in \mathcal{ECU}_{bC} \wedge \widehat{\kappa}_{\text{msgid}}(a) = \text{fmsg} \Rightarrow \widehat{\kappa}_{\text{ecucnt}}(a) = \widehat{\kappa}_{\text{cnt}}(a) = \widehat{\kappa}_{\text{prevcnt}}(a) - 1$.

Prop. B11. All honest ECUs, when reading a message, check the message's MAC, independently of whether or not they accept it. In case of a functional message, this involves the ECU's local counter, more concretely the counter value used by the ECU in its previous action decremented by 1. If such a check succeeds which is a necessary condition for the error frame being set to no, this value is the one that was used to generate the message's MAC. Note that this assumes that an attacker that owns the correct counter and a MAC cannot guess the corresponding message.

Let $s \in \text{alph}(\omega)$ with $\widehat{\kappa}_{\text{aname}}(s) = \text{send}_{bC}$ and $v \in \Phi(s, W_{bCnt})$. Then the following holds:

$\forall b \in \{a \in \text{alph}(v) | \widehat{\kappa}_{\text{aname}}(a) \in \{\text{read}_{bC}, \text{recv}_{bC}\} \wedge \widehat{\kappa}_{\text{ecu}}(a) \in \mathcal{ECU}_{bC} \wedge \widehat{\kappa}_{\text{msgid}}(a) = \text{fmsg}\}$: $\widehat{\kappa}_{\text{errorFrame}}(b) = \text{no} \Rightarrow \widehat{\kappa}_{\text{cnt}}(b) = \widehat{\kappa}_{\text{ecucnt}}(b) = \widehat{\kappa}_{\text{prevcnt}}(a) - 1$.

Note that in our very simple model with only one type of functional message, a successful check indicated by errorFrame = no actually results into a recv_{bC} action. However, considering also read_{bC} actions with errorFrame = no allows to extend the model with respect to more types of

functional messages without having to change the assumptions.

Prop. B12. As explained in Section 3, if an ECU's checks concerning for example a message's MAC fails and it therefore writes an error-frame, all other ECUs join in and write an error-frame as well, no matter whether or not their checks failed. Hence all read_{bC} and recv_{bC} actions induced by a specific send_{bC} action have the same value for the parameter errorFrame. Since a message is only received and accepted if all checks have been successful, the error frame of a recv_{bC} action is always set to no. Formally:

Let $s \in \text{alph}(\omega)$ with $\widehat{\kappa}_{\text{aname}}(s) = \text{send}_{bC}$ and $\nu \in \Phi(s, W_{bCnt})$. Let further $R(\nu) \coloneqq \{b \in \text{alph}(\nu) | \widehat{\kappa}_{\text{aname}}(b) \in \{\text{read}_{bC}, \text{recv}_{bC}\}\}$ denote the read_{bC} and recv_{bC} actions in ν. Then for all $b_i, b_j \in R(\nu)$ the following holds:

$\widehat{\kappa}_{\text{errorFrame}}(b_i) = \widehat{\kappa}_{\text{errorFrame}}(b_j)$ and $\widehat{\kappa}_{\text{aname}}(b_i) = \text{recv}_{bC} \Rightarrow \widehat{\kappa}_{\text{errorFrame}}(b_i) = \text{no}$.

Prop. B13 (in contrast to Prop.A10). When an honest ECU reads a message, it always decrements its previously used bCnt value by 1 and uses the result as its new local bCnt value. This behavior is independent of whether or not its checks fail, i.e. independent of the errorFrame value. Formally:

$\forall a \in \text{alph}(\omega)$: $\widehat{\kappa}_{\text{aname}}(a) = \text{read}_{bC} \wedge \widehat{\kappa}_{\text{ecu}}(a) \in \mathcal{E}CU_{bC} \Rightarrow \widehat{\kappa}_{\text{ecucnt}}(a) = \widehat{\kappa}_{\text{prevcnt}}(a) - 1$.

Prop. B14. In a phase class $\Phi(s, W_{bCnt})$ with $\widehat{\kappa}_{\text{aname}}(s) = \text{send}_{bC}$ (i.e. a phase class that starts with a specific send_{bC} action and ends with the next send_{bC} action, see Definition 9), all honest ECUs including the sender either read or receive the message or perform a loseCnt action. They do not perform any other action.

Let $s \in \Sigma_{bCnt}$ with $\widehat{\kappa}_{\text{aname}}(s) = \text{send}_{bC}$ and ν maximal in $\Phi(s, W_{bCnt})$. Then for all $\text{ecu} \in \mathcal{E}CU_{bC}$ exists exactly one $c \in \text{alph}(\nu)$ such that $\widehat{\kappa}_{\text{ecu}}(c) = \text{ecu}$ and $\widehat{\kappa}_{\text{aname}}(c) \in \{\text{read}_{bC}, \text{recv}_{bC}, \text{loseCnt}_{bC}\}$.

Prop. B15 (analog to Prop.A6). In every phase class $\Phi(s, W_{bCnt})$ with $\widehat{\kappa}_{\text{aname}}(s) = \text{send}_{bC}$ there is an honest ECU owning the correct counter and performing a read or recv_{bC} action in this phase class, but no loseCnt action. Here, owning the correct counter means that the ECU has used and stored the correct counter value in its previous action and can thus use it in the next action.

Let $s \in \text{alph}(\omega)$ with $\widehat{\kappa}_{\text{aname}}(s) = \text{send}_{bC}$. Then for all ν maximal in $\Phi(s, W_{bCnt})$ exists an action $b \in alph(\nu)$ with $\widehat{\kappa}_{\text{aname}}(b) \in \{\text{read}, \text{recv}\}, \text{ECU}^*(s) \coloneqq \widehat{\kappa}_{\text{ecu}}(b) \in \mathcal{E}CU_{bC}$ and $\widehat{\kappa}_{\text{prevcnt}}(b) = \text{corCnt}_{bC}(b, \omega) + 1$.

Prop. B16 (analog to Prop.A12). If an ECU is not synchronized at a specific action, i.e. does not use the correct counter relevant for this action, it must have performed an action loseCnt_{bC} before. Note that using a counter value refers to the parameter prevcnt.

$\forall a \in \text{alph}(\omega)$: $\widehat{\kappa}_{\text{prevcnt}}(a) - 1 \neq \text{corCnt}_{bC}(a, \omega) \Rightarrow \exists a' \in \text{alph}(\omega)$: $\widehat{\kappa}_{\text{aname}}(a') = \text{loseCnt}_{bC} \wedge \widehat{\kappa}_{\text{ecu}}(a') = \widehat{\kappa}_{\text{ecu}}(a) \wedge \text{actCnt}(a', \omega) < \text{actCnt}(a, \omega)$

Prop. B17 (analog to Prop.A11). As explained in Prop.B5, memory failures never result into decrease of the counter value stored by an ECU. This implies that an action loseCnt_{bC} performed by an honest ECU resets the ECU's counter to a value bigger than the correct one. For formal reasons we assign a counter value higher than the maximal value the system starts with to a loseCnt action if it is the first action of an action sequence.

$\forall a \in \text{alph}(\omega)$:

(1) $a \neq \text{pre}_1(\omega) \wedge \widehat{\kappa}_{\text{aname}}(a) = \text{loseCnt} \wedge \widehat{\kappa}_{\text{ecu}}(a) \in \mathcal{E}CU_{bC} \Rightarrow \widehat{\kappa}_{\text{ecucnt}}(a) > \text{corCnt}_{bC}(a, \omega)$.

(2) $a = \text{pre}_1(\omega) \wedge \widehat{\kappa}_{\text{aname}}(a) = \text{loseCnt} \wedge \widehat{\kappa}_{\text{ecu}}(a) \in \mathcal{E}CU_{bC} \Rightarrow \widehat{\kappa}_{\text{ecucnt}}(a) = bCnt_{\max} + 1$

Analogously to Section 5.1.2 we model immediacy and non-repeatability by the phase class $\Phi(\sigma(b), W_{bCnt})$ as defined in Definition 9 determined by a fixed but arbitrary recv_{bC} action b. In the following section we will show that BusCnt satisfies both properties.

6.2. Formal Proof of bCnt System. The idea of the BusCnt system is that counter values included in the MACs of sent messages are strictly monotonically decreasing (instead of strictly monotonically increasing as in the GenCnt system). However, in contrast to the GenCnt system, in BusCnt each send action inevitably induces a read_{bC} or recv_{bC} action and thus a decrement of the counter, no matter whether or not a check failed. In case a system does not suffer any anomalies (i.e. all actors act correctly and counter value change is never caused by physical irregularities), the counter used by the ECUs is always the correct one. Lemma 2 will show how it is determined. For its proof we need the following technical Lemma:

Lemma 1. *Let* $\omega \in W^{cor}_{bCnt} \coloneqq \{\omega \in W_{bCnt} | \widehat{\kappa}_{ecu}(a) \in \mathcal{E}CU_{bC} \wedge \widehat{\kappa}_{aname}(a) \neq loseCnt \text{ for all } a \in alph(\omega)\}$. *Let further* $S(\omega) \coloneqq \{a \in alph(\omega) | \widehat{\kappa}_{aname}(a) = send_{bC}\}$ *with* $actCnt(s_i, \omega) < actCnt(s_{(i+k)}, \omega)$ *for all* $s_i, s_{i+k} \in S(\omega)(i, k \in \mathbb{N})$. *Then for all* $b \in alph(\omega)$ *with* $\widehat{\kappa}_{aname}(b) \in \{read_{bC}, recv_{bC}\}$ *the following holds:*

1. $\widehat{\kappa}_{msgid}(b) = fmsg \Rightarrow \widehat{\kappa}_{ecucnt}(b) = \widehat{\kappa}_{cnt}(b)$
2. $\widehat{\kappa}_{msgid}(b) = sync \Rightarrow \widehat{\kappa}_{ecucnt}(b) = \widehat{\kappa}_{msg}(b)$

Proof 2. If $\widehat{\kappa}_{\text{aname}}(b) = \text{recv}_{bC}$, the assertions follow directly by Prop.B10 and Prop.B12 together with Prop.B8, respectively. Let now $\widehat{\kappa}_{\text{aname}}(b) = \text{read}_{bC}$. We show the assertions of this case by induction over the number of consecutive phase classes $\Phi(s_i, W^{\text{cor}}_{bCnt})$.

Induction basis: $i = 1$. Consider $\nu \in \Phi(s_1, W^{\text{cor}}_{bCnt})$ and $b \in \text{alph}(\nu)$. Let $\widehat{\kappa}_{\text{msgid}}(b) = \text{fmsg}$ and assume $\widehat{\kappa}_{\text{ecu}}(s_1) = \widehat{\kappa}_{\text{ecu}}(b)$. Then Prop.B13 and Prop.B4 imply

$\widehat{\kappa}_{\text{ecucnt}}(b) = \widehat{\kappa}_{\text{prevcnt}}(b) - 1 = \widehat{\kappa}_{\text{ecucnt}}(s_1) - 1$. Further, by Prop.B1, $\widehat{\kappa}_{\text{cnt}}(b) = \widehat{\kappa}_{\text{cnt}}(s_1)$. Now by Prop.B9, $\widehat{\kappa}_{\text{cnt}}(s_1) = \widehat{\kappa}_{\text{prevcnt}}(s_1) - 1 = \widehat{\kappa}_{\text{ecucnt}}(s_1) - 1$. Together this leads to $\widehat{\kappa}_{\text{cnt}}(b) = \widehat{\kappa}_{\text{ecucnt}}(s_1) - 1 = \widehat{\kappa}_{\text{ecucnt}}(b)$.

Assume now $\widehat{\kappa}_{\text{ecu}}(b) \neq \widehat{\kappa}_{\text{ecu}}(s_1)$. Since Prop.B1 requires a send action before any read_{bC} or recv_{bC} action and since s_1 is the first send action in ω, there is no other action in ω before s_1. Prop.B14 implies that b is the first $\widehat{\kappa}_{\text{ecu}}(b) \in \Omega\widehat{\kappa}_{\text{prevcnt}}(b) = bCnt_{\max}$ and thus Prop.B13 implies $\widehat{\kappa}_{\text{ecucnt}}(b) = \widehat{\kappa}_{\text{prevcnt}}(b) - 1 = bCnt_{\max} - 1$. s_1 being the first action in ω, it is the first action of $\widehat{\kappa}_{\text{ecu}}(s_1)$ as well and Prop.B4 implies $\widehat{\kappa}_{\text{prevcnt}}(s_1) = bCnt_{\max}$. By Prop.B9 it follows $\widehat{\kappa}_{\text{cnt}}(s_1) = \widehat{\kappa}_{\text{prevcnt}}(s_1) - 1 = bCnt_{\max} - 1$. Further, by Prop.B1, $\widehat{\kappa}_{\text{cnt}}(b) = \widehat{\kappa}_{\text{cnt}}(s_1)$. Together we can conclude $\widehat{\kappa}_{\text{cnt}}(s_1) = \widehat{\kappa}_{\text{cnt}}(s_1) = bCnt_{\max} - 1 = \widehat{\kappa}_{\text{ecucnt}}(b)$.

In case $\widehat{\kappa}_{\text{msgid}}(b) = \text{sync}$, we can argue analogously by replacing every occurrence of $\widehat{\kappa}_{\text{cnt}}$ by $\widehat{\kappa}_{\text{msg}}$ and applying Prop.B6 instead of Prop.B9.

Induction hypothesis: For $v \in \Phi(s_i, W^{\text{cor}}_{bCnt})$ and $b \in \text{alph}(v)$, let assertions 1 and 2 hold.

Induction step: Consider $v \in \Phi(s_{i+1}, W^{\text{cor}}_{bCnt}), b \in alph(v)$ with $\widehat{\kappa}_{\text{aname}}(b) \in \{\text{read}_{bC}, \text{recv}_{bC}\}$ and $\widehat{\kappa}_{\text{msgid}}(b) = \text{fmsg}$. First we again assume $\widehat{\kappa}_{\text{ecu}}(b) = \widehat{\kappa}_{\text{ecu}}(s_{i+1})$. b being a read_{bC} action, Prop.B13 implies $\widehat{\kappa}_{\text{ecucnt}}(b) = \widehat{\kappa}_{\text{prevcnt}}(b) - 1$ which by Prop.B4 and Prop.B14 is equal to $\widehat{\kappa}_{\text{ecucnt}}(s_{i+1}) - 1$. Further, by Prop.B9, $\widehat{\kappa}_{\text{cnt}}(s_{i+1}) = \widehat{\kappa}_{\text{prevcnt}}(s_{i+1}) - 1$ and $\widehat{\kappa}_{\text{ecucnt}}(s_{i+1}) = \widehat{\kappa}_{\text{prevcnt}}(s_{i+1})$. Since Prop.B1 implies $\widehat{\kappa}_{\text{cnt}}(s_{i+1}) = \widehat{\kappa}_{\text{cnt}}(s_{i+1})$, it follows $\widehat{\kappa}_{\text{cnt}}(b) = \widehat{\kappa}_{\text{prevcnt}}(s_{i+1}) - 1 = \widehat{\kappa}_{\text{ecucnt}}(s_{i+1}) - 1 = \widehat{\kappa}_{\text{ecucnt}}(b)$.

Assume now $\widehat{\kappa}_{\text{ecu}}(b) \neq \widehat{\kappa}_{\text{ecu}}(s_{i+1})$. As above, Prop.B13 implies $\widehat{\kappa}_{\text{ecucnt}}(b) = \widehat{\kappa}_{\text{prevcnt}}(b) - 1$. Since by Prop.B14 all ECUs perform a read_{bC} or recv_{bC} action in the previous phase class $\Phi(s_i, W^{\text{cor}}_{bCnt})$ (loseCnt_{bC} actions are excluded by definition), this holds in particular for $\widehat{\kappa}_{\text{ecu}}(b)$ and $\widehat{\kappa}_{\text{ecu}}(s_{i+1})$. Hence for all maximal v' in $\Phi(s_i, W^{\text{cor}}_{bCnt})$ there exist read_{bC} or recv_{bC} actions $b' \in \text{alph}(v')$ performed by $\widehat{\kappa}_{\text{ecu}}(b)$ and $a \in \text{alph}(v')$ performed by $\widehat{\kappa}_{\text{ecu}}(s_{i+1})$, being the previous actions of $\widehat{\kappa}_{\text{ecu}}(b)$ and $\widehat{\kappa}_{\text{ecu}}(s_{i+1})$, respectively. Prop.B4 implies $\widehat{\kappa}_{\text{ecucnt}}(b) = \widehat{\kappa}_{\text{prevcnt}}(b) - 1 = \widehat{\kappa}_{\text{ecucnt}}(b') - 1$ which by induction hypothesis is equal to $\widehat{\kappa}_{\text{cnt}}(b') - 1$. This in turn is equal to $\widehat{\kappa}_{\text{cnt}}(s_i) - 1 = \widehat{\kappa}_{\text{cnt}}(a) - 1$ by Prop.B1. Again by induction hypothesis, the latter expression is equal to $\widehat{\kappa}_{\text{ecucnt}}(a) - 1$. Prop.B4 implies equality to $\widehat{\kappa}_{\text{prevcnt}}(s_{i+1}) - 1$ which by Prop.B9 is equal to $\widehat{\kappa}_{\text{cnt}}(s_{i+1})$. Prop.B1 finally implies equality to $\widehat{\kappa}_{\text{cnt}}(b)$, hence $\widehat{\kappa}_{\text{cnt}}(b) = \widehat{\kappa}_{\text{ecucnt}}(b)$.

Again, in case $\widehat{\kappa}_{\text{msgid}}(b) = \text{sync}$, the analogous proof is achieved by replacing every occurrence of $\widehat{\kappa}_{\text{cnt}}$ by $\widehat{\kappa}_{\text{msg}}$ and applying Prop.B6 instead of Prop.B9. □

Lemma 2. *Let W^{cor}_{bCnt} and $S(\omega)$ as defined in Lemma 1. Let further $\omega \in W^{cor}_{bCnt}$ and $a \in alph(\omega)$. Then the following holds:*

1. $\widehat{\kappa}_{msgid}(a) = fmsg \Rightarrow \widehat{\kappa}_{cnt}(a) = bCnt_{\max} - card(\{s \in S(\omega) | actCnt(s,\omega) \leq actCnt(a,\omega)\}) \geq 0$
2. $\widehat{\kappa}_{msgid}(a) = sync \Rightarrow \widehat{\kappa}_{msg}(a) = bCnt_{\max} - card(\{s \in S(\omega) | actCnt(s,\omega) \leq actCnt(a,\omega)\}) \geq 0$

Further, for all $s_{i-1}, s_i \in S(\omega)$ (i.e. with $actCnt(s_{i-1}, \omega) < actCnt(s_i, \omega)$ and $i \in \mathbb{N}, i \geq 2$) holds

(i) $\widehat{\kappa}_{msgid}(s_{i-1}) = \widehat{\kappa}_{msgid}(s_i) = fmsg \Rightarrow \widehat{\kappa}_{cnt}(s_i) = \widehat{\kappa}_{cnt}(s_{i-1}) - 1$

(ii) $\widehat{\kappa}_{msgid}(s_{i-1}) = \widehat{\kappa}_{msgid}(s_i) = sync \Rightarrow \widehat{\kappa}_{msg}(s_i) = \widehat{\kappa}_{msg}(s_{i-1}) - 1$

(iii) $\widehat{\kappa}_{msgid}(s_{i-1}) = fmsg \wedge \widehat{\kappa}_{msgid}(s_i) = sync \Rightarrow \widehat{\kappa}_{msg}(s_i) = \widehat{\kappa}_{cnt}(s_{i-1}) - 1$

(iv) $\widehat{\kappa}_{msgid}(s_{i-1}) = sync \wedge \widehat{\kappa}_{msgid}(s_i) = fmsg \Rightarrow \widehat{\kappa}_{cnt}(s_i) = \widehat{\kappa}_{msg}(s_{i-1}) - 1$

Note that item 1 implies that the parameter cnt of actions concerning a functional message never reaches the value 0 unless there occur no more send_{bC} actions after the action a. The analogous statement holds for the parameter msg of actions concerning synchronization messages.

Proof 3. We prove assertions 1 and 2 by induction over the length $l \in \mathbb{N}$ of a word $\omega \in W^{\text{cor}}_{bCnt}$.

Induction basis: $l = 1$, i.e. $\omega = a_1$. Since Prop.B1 requires a send_{bC} action before any read_{bC} or recv_{bC} action, a_1 cannot be a read_{bC} or recv_{bC} action. Since further by definition ω does not contain any loseCnt action, $\widehat{\kappa}_{\text{aname}}(a_1) = \text{send}_{bC}$. Prop.B4 implies $\widehat{\kappa}_{\text{prevcnt}}(a_1) = bCnt_{\max}$. If $\widehat{\kappa}_{\text{msgid}}(a_1) = \text{fmsg}$, by Prop.B9 it follows $\widehat{\kappa}_{\text{cnt}}(a_1) = \widehat{\kappa}_{\text{prevcnt}}(a_1) - 1 = bCnt_{\max} - 1 = bCnt_{\max} - \text{card}(\{s \in S(a_1) | \text{actCnt}(s, a_1) \leq \text{actCnt}(a_1, a_1)\})$. Further, by Prop.B5 $\widehat{\kappa}_{\text{prevcnt}}(a_1) = bCnt_{\max} > 0$ which implies $\widehat{\kappa}_{\text{prevcnt}}(a_1) - 1 = bCnt_{\max} - 1 \geq 0$. Thus item 1 holds for $\omega = a_1$ containing a functional message. If on the other hand $\widehat{\kappa}_{\text{msgid}}(a_1) = \text{sync}$, by Prop.B6 it follows $\widehat{\kappa}_{\text{msg}}(a_1) = \widehat{\kappa}_{\text{prevcnt}}(a_1) - 1$ which as above implies the assertion, thus item 2 holds for $\omega = a_1$.

Induction hypothesis: Let $\omega_i = a_1 \ldots a_i$ ($i \geq 2, i \in \mathbb{N}$). Then for all $a \in \text{alph}(\omega_i)$ holds:

(1) $\widehat{\kappa}_{\text{msgid}}(a) = \text{fmsg} \Rightarrow \widehat{\kappa}_{\text{cnt}}(a) = bCnt_{\max} - \text{card}(\{s \in S(\omega_i) | \text{actCnt}(s, \omega_i) \leq \text{actCnt}(a, \omega_i)\}) \geq 0$

(2) $\widehat{\kappa}_{\text{msgid}}(a) = \text{sync} \Rightarrow \widehat{\kappa}_{\text{msg}}(a) = bCnt_{\max} - \text{card}(\{s \in S(\omega_i) | \text{actCnt}(s, \omega_i) \leq \text{actCnt}(a, \omega_i)\}) \geq 0$

Induction step: Consider $\omega_{i+1} := a_1 \ldots a_i a_{i+1}$.

(1) Assume $\widehat{\kappa}_{\text{aname}}(a_{i+1}) = \text{send}_{bC}$. By Prop.B14 and the fact that ω_{i+1} does not contain any loseCnt_{bC} actions, it follows that a_i is a recv_{bC} or read_{bC} action. This in turn is preceded by a send_{bC} action $\sigma(a_i)$ (see Prop.B1). So we have $\omega_{i+1} = a_1 \ldots \sigma(a_i) \ldots a_i a_{i+1}$ ($\sigma(a_i)$ may or may not be equal to a_1). Let v_i a maximal word in $\Phi(\sigma(a_i), W^{\text{cor}}_{bCnt})$, i.e. v_i starts with $\sigma(a_i)$ and ends with a_{i+1}, and all other actions in between are read_{bC} and recv_{bC} actions, the last one being a_i. By Prop.B1, for all these read_{bC} and recv_{bC} actions $b \in \text{alph}(v_i)$ holds $\widehat{\kappa}_{\text{cnt}}(b) = \widehat{\kappa}_{\text{cnt}}(\sigma(a_i)) = \widehat{\kappa}_{\text{cnt}}(a_i)$. Since all ECUs perform exactly one $read_{bC}$ or recv_{bC} action in v_i, there is exactly one recv_{bC} or read_{bC} action $b^* \in \text{alph}(v_i)$ performed by $\text{ecu}^* := \widehat{\kappa}_{\text{ecu}}(a_{i+1})$, being the last action performed by ecu^* before a_{i+1}.

(a) Assume $\widehat{\kappa}_{\text{msgid}}(a_{i+1}) = \text{fmsg}$. If $\widehat{\kappa}_{\text{msgid}}(b^*) = \text{fmsg}$ as well, Lemma 1 implies $\widehat{\kappa}_{\text{ecucnt}}(b^*) = \widehat{\kappa}_{\text{cnt}}(b^*)$.

Since a_{i+1} contains a functional message, Prop.B9 implies $\widehat{\kappa}_{\text{cnt}}(a_{i+1}) = \widehat{\kappa}_{\text{prevcnt}}a_{(i+1)} - 1 = \widehat{\kappa}_{\text{ecucnt}}(b^*) - 1 = \widehat{\kappa}_{\text{cnt}}(b^*) - 1$. By Prop.B1 this is equal to $\widehat{\kappa}_{\text{cnt}}(\sigma(a_i)) - 1 = \widehat{\kappa}_{\text{cnt}}(a_i) - 1$. Prop.B1 also implies $\widehat{\kappa}_{\text{msgid}}(a_i) = \widehat{\kappa}_{\text{msgid}}(b^*) = \text{fmsg}$, hence by induction hypothesis it follows $\widehat{\kappa}_{\text{cnt}}(a_{i+1}) = \widehat{\kappa}_{\text{cnt}}(a_i) - 1 = bCnt_{\max} - \text{card}(\{s \in S(\omega_{i+1}) | \text{actCnt}(s, \omega_{i+1}) \leq \text{actCnt}(a_i, \omega_{i+1})\}) - 1$. Finally, since a_{i+1} is the send_{bC} action directly following a_i, $\widehat{\kappa}_{\text{cnt}}(a_{i+1}) = bCnt_{\max} - \text{card}\{s \in S(\omega_{i+1}) | \text{actCnt}(s, \omega_{i+1}) \leq \text{actCnt}(a_{i+1}, \omega_{i+1})\}$. Further, since $\widehat{\kappa}_{\text{cnt}}(a_{i+1}) = \widehat{\kappa}_{\text{prevcnt}}(a_{i+1}) - 1$ and by Prop.B5 $\widehat{\kappa}_{\text{prevcnt}}(a_{i+1}) > 0$, it follows $\widehat{\kappa}_{\text{cnt}}(a_{i+1}) \geq 0$.

If on the other hand $\widehat{\kappa}_{\text{msgid}}(b^*) = \text{sync}$, Lemma 1 implies $\widehat{\kappa}_{\text{ecucnt}}(b^*) = \widehat{\kappa}_{\text{msg}}(b^*)$. Using Prop.B9, we can then deduce $\widehat{\kappa}_{\text{cnt}}(a_{i+1}) = \widehat{\kappa}_{\text{ecucnt}}(b^*) - 1 = \widehat{\kappa}_{\text{msg}}(b^*) - 1$ which by Prop.B1 is equal to $\widehat{\kappa}_{\text{msg}}(a_i) - 1$. As above, the assertion follows.

(b) Assume $\widehat{\kappa}_{\text{msgid}}(a_{i+1}) = \text{sync}$. In this case we can use Prop.B6 to deduce $\widehat{\kappa}_{\text{msg}}(a_{i+1}) = \widehat{\kappa}_{\text{prevcnt}}(a_{i+1}) - 1$. If $\widehat{\kappa}_{\text{msgid}}(b^*) = \text{fmsg}$, the rest of the proof is identical to the case where this is combined with a functional message of a_{i+1}, if $\widehat{\kappa}_{\text{msgid}}(b^*) = \text{sync}$, the arguments regarding a_{i+1} being a synchronization message apply. This ends the proof for the case $\widehat{\kappa}_{\text{aname}}(a_{i+1}) = \text{send}_{bC}$.

(2) Assume $\widehat{\kappa}_{\text{aname}}(a_{i+1}) \in \{\text{read}_{bC}, \text{recv}_{bC}\}$. Then Prop.B1 implies that there is an action $\sigma(a_{i+1})$ which is either equal to a_i or occurs before a_i.

(a) Let $\sigma(a_{i+1}) = a_i$. By Prop.B1 it follows $\widehat{\kappa}_{\text{cnt}}(a_{i+1}) = \widehat{\kappa}_{\text{cnt}}(a_i), \widehat{\kappa}_{\text{msg}}(a_{i+1}) = \widehat{\kappa}_{\text{msg}}(a_i)$ and $\widehat{\kappa}_{\text{msgid}}(a_{i+1}) = \widehat{\kappa}_{\text{msgid}}(a_i)$. Since a_i is the last send action before the $\text{recv}_{bC}/\text{read}_{bC}$ action a_{i+1}, the number of send actions before these two actions including a_i is identical, i.e. $\text{card}(\{s \in S(\omega_{i+1}) | \text{actCnt}(s, \omega_{i+1}) \leq \text{actCnt}(a_i, \omega_{i+1})\}) = \text{card}(\{s \in S(\omega_{i+1}) | \text{actCnt}(s, \omega_{i+1}) \leq \text{actCnt}(a_{i+1}, \omega_{i+1})\})$. If $\widehat{\kappa}_{\text{msgid}}(a_i) = \text{fmsg}$, the induction hypothesis implies $0 \leq \widehat{\kappa}_{\text{cnt}}(a_i) = bCnt_{\max} - \text{card}(\{s \in S(\omega_{i+1}) | \text{actCnt}(s, \omega_{i+1}) \leq \text{actCnt}(a_i, \omega_{i+1})\}) = bCnt_{\max} - \text{card}(\{s \in S(\omega_{i+1}) | \text{actCnt}(s, \omega_{i+1}) \leq \text{actCnt}(a_{i+1}, \omega_{i+1})\}) = \widehat{\kappa}_{\text{cnt}}(a_{i+1})$. If $\widehat{\kappa}_{\text{msgid}}(a_i) = \text{sync}$, the induction hypothesis implies $0 \leq \widehat{\kappa}_{\text{msg}}(a_i) = bCnt_{\max} - \text{card}(\{s \in S(\omega_{i+1}) | \text{actCnt}(s, \omega_{i+1}) \leq \text{actCnt}(a_i, \omega_{i+1})\}) = bCnt_{\max} - \text{card}(\{s \in S(\omega_{i+1}) | \text{actCnt}(s, \omega_{i+1}) \leq \text{actCnt}(a_{i+1}, \omega_{i+1})\}) = \widehat{\kappa}_{\text{msg}}(a_{i+1})$.

(b) Let $\sigma(a_{i+1}) =: a'$ with $\text{actCnt}(a', \omega_{i+1}) > \text{actCnt}(a_i, \omega_{i+1})$. Then Prop.B14 and the fact that ω_{i+1} does not contain any loseCnt_{bC} actions implies that a_i is a read_{bC} or recv_{bC} action. Again, by Prop.B1 it follows $\widehat{\kappa}_{\text{cnt}}(a_i) = \widehat{\kappa}_{\text{cnt}}(\sigma(a_i)) = \widehat{\kappa}_{\text{cnt}}(\sigma(a_{i+1})) = \widehat{\kappa}_{\text{cnt}}(a_{i+1})$ and the equivalent equations for the parameters msg and msgid. Now we can again conclude that the number of send actions before a_i is identical to those before a_{i+1} and as above, by induction hypothesis for the cases $\widehat{\kappa}_{\text{msgid}}(a_i) = \text{fmsgid}$ and $\widehat{\kappa}_{\text{msgid}}(a_i) = \text{sync}$, respectively, it follows the assertion. This concludes the proof of assertions 1 and 2.

We now prove assertions (i)–(iv) of the Lemma. So assume the first statement holds regarding functional messages. Then it holds in particular for two consecutive send_{bC} actions s_{i-1} and s_i in a word $\omega \in W^{\text{cor}}_{bCnt}$, each sending a functional message. Then $\widehat{\kappa}_{\text{cnt}}(s_i) = bCnt_{\max} - \text{card}(\{s \in S(\omega) | \text{actCnt}(s, \omega) \leq \text{actCnt}(s_i, \omega)\}) = bCnt_{\max} - (\text{card}(\{s \in S(\omega) | \text{actCnt}(s, \omega) \leq \text{actCnt}(s_{i-1}, \omega)\}) + 1) = bCnt_{\max} - \text{card}(\{s \in S(\omega) | \text{actCnt}(s, \omega) \leq \text{actCnt}(s_{i-1}, \omega)\}) - 1 = \widehat{\kappa}_{\text{cnt}}(s_{i-1}) - 1$. The only difference between a send_{bC} action s containing a synchronization message and one containing a functional message is that the value $bCnt_{\max}$ minus the cardinality of send_{bC} actions happening before s and including s is assigned to the counter of s in the first case and to the message of s in the second case. Hence, the respective statements for all other combinations of synchronization and functional send_{bC} actions can be shown analogously. □

The above Lemma uses properties that describe the behavior of honest agents when sending, receiving or reading a message and shows the resulting counter value. This behavior does not depend on whether or not any involved send action is performed by an honest ECU. Hence the counter value included in actions of sequences in W^{cor}_{bCnt} can be considered the correct one for actions in W_{bCnt}.

Definition 10. *For $\omega \in W_{bCnt}$ and $a \in \text{alph}(\omega)$ we define*

$$\text{corCnt}_{bC}(a, \omega) := bCnt_{\max} - \text{card}(\{s \in S(\omega) | \text{actCnt}(s, \omega) \leq \text{actCnt}(a, \omega)\}). \tag{9}$$

We now consider W_{bCnt} again and first show a Lemma whose important statement is that a synchronization message received by an ECU never contains a counter smaller than the correct one for this action. In general the Lemma states that an honest ECU owning the correct counter before processing a read_{bC} or recv_{bC} action (denoted by the parameter prevcnt of the action), also owns the correct counter afterwards (denoted by the parameter ecucnt).

Lemma 3. *Let $\omega \in W_{bCnt}$ and $a \in alph(\omega)$. Then the following holds:*

$$\widehat{\kappa}_{\text{aname}}(a) \in \{\text{read}_{bC}, \text{recv}_{bC}\} \wedge \widehat{\kappa}_{\text{prevcnt}}(a) = \text{corCnt}_{bC}(a, \omega) + 1 \Rightarrow \widehat{\kappa}_{\text{ecucnt}}(a) = \text{corCnt}_{bC}(a, \omega). \tag{10}$$

Proof 4. Assume $\widehat{\kappa}_{\text{aname}}(a) = \text{read}_{bC}$. Then by Prop.B13 $\widehat{\kappa}_{\text{ecucnt}}(a) = \widehat{\kappa}_{\text{prevcnt}}(a) - 1 = \text{corCnt}_{bC}(a, \omega) + 1 - 1 = \text{corCnt}_{bC}(a, \omega)$. The same follows by Prop.B10 for a being a recv_{bC} action of a functional message. So let $\widehat{\kappa}_{\text{aname}}(a) = \text{recv}_{bC}$ and $\widehat{\kappa}_{\text{msgid}}(a) = \text{sync}$. Prop.B12 implies $\widehat{\kappa}_{\text{errorFrame}}(a) = \text{no}$ and thus by Prop.B8 it follows $\widehat{\kappa}_{\text{ecucnt}}(a) \leq \widehat{\kappa}_{\text{prevcnt}}(a) - 1 = \text{corCnt}_{bC}(a, \omega) + 1 - 1$. In case of $\widehat{\kappa}_{\text{ecucnt}}(a) = \widehat{\kappa}_{\text{prevcnt}}(a) - 1$ the assertion holds. So let $\widehat{\kappa}_{\text{ecucnt}}(a) < \widehat{\kappa}_{\text{prevcnt}}(a) - 1$, i.e. $\widehat{\kappa}_{\text{ecucnt}}(a) < \text{corCnt}_{bC}(a, \omega)$. Prop.B16 implies that before a, $\widehat{\kappa}_{\text{ecu}}(a)$ performs a loseCnt action a' and by Prop.B17, if $a' \neq \text{pre}_1(\omega)$,

it follows $\widehat{\kappa}_{\text{ecucnt}}(a') > \text{corCnt}_{bC}(a', \omega)$. For simplicity we assume that a' is the last action of $\widehat{\kappa}_{\text{ecu}}(a)$ in ω before a. By Prop.B14, every ECU performs only one action per phase class, hence a' happens in the phase class $\Phi(s', W_{bCnt})$ that ends with $\sigma(a)$, i.e. in the phase class directly occuring before the one including a. Lemma 2 implies that $\text{corCnt}_{bC}(a', \omega) = \text{corCnt}_{bC}(a, \omega) + 1$. So we have $\text{corCnt}_{bC}(a, \omega) + 1 = \widehat{\kappa}_{\text{prevcnt}}(a) = \widehat{\kappa}_{\text{ecucnt}}(a') > \text{corCnt}_{bC}(a', \omega) = \text{corCnt}_{bC}(a, \omega) + 1$. This constitutes a contradiction, hence $\widehat{\kappa}_{\text{ecucnt}}(a) = \text{corCnt}_{bC}(a, \omega)$ holds. In case a' is not the previous action of $\widehat{\kappa}_{\text{ecu}}(a)$, we can argue analogously.

Let $a' = \text{pre}_1(\omega)$. Since a' is the first action and not a send_{bC} action, Lemma 2 implies $\text{corCnt}_{bC}(a', \omega) = bCnt_{\max}$. As above, we have $\text{corCnt}_{bC}(a, \omega) + 1 = \widehat{\kappa}_{\text{prevcnt}}(a) = \widehat{\kappa}_{\text{ecucnt}}(a')$ which by Prop.B17 is equal to $bCnt_{\max} + 1 = \text{corCnt}_{bC}(a', \omega) + 1$. This implies $\text{corCnt}_{bC}(a, \omega) = \text{corCnt}_{bC}(a', \omega)$. However, since $\sigma(a)$ happens before the recv_{bC} action a (and after the loseCnt_{bC} action a'), by Lemma 2 $\text{corCnt}_{bC}(a) = bCnt_{\max} - 1 \neq bCnt_{\max} = \text{corCnt}_{bC}(a', \omega)$. So again this constitutes a contradiction, hence $\widehat{\kappa}_{\text{ecucnt}}(a) = \text{corCnt}_{bC}(a, \omega)$ always holds. □

We can now prove our main Theorem. As in Section 5.2, the property we want to prove is that whenever an honest ECU receives and accepts a message (action b), the send_{bC} action $\sigma(b)$ having triggered b and starting the phase class determined by b must have authentically for the ECU been performed by an agent being member of the same group, and the message must contain the correct counter. In contrast to the proof regarding the GenCnt system, for the BusCnt system (formally denoted by B_{bCnt}) we can show that this property is always satisfied, i.e. that both immediacy and non-repeatability hold.

Theorem 2. *Let* $\omega \in B_{bCnt}$ *and* $b := (recv_{bC}, ecu, ecukey, ecucnt, prevcnt, bus, mackey, msgid, msg, cnt) \in alph(\omega)$ *with* $ecu \in \mathcal{ECU}_{bC}$. *Then the following property holds:*

$$\text{authWiPhase}\{(\text{send}, \text{ecu}', \text{ecukey}', \text{ecucnt}', \text{prevcnt}', \text{bus}, \text{mackey}, \text{msgid}, \text{msg}, \text{cnt}) | ecu' \in \mathcal{ECU}_{bC}\}, b, \text{ecu}, \Phi(\sigma(b), W_{bCnt}.) \tag{11}$$

Proof 5. Analogously to Section 5.2, without loss of generality we assume $\text{ECU}_1^{bC} \in \mathcal{ECU}_{bC}$ to perform a receive action $b := (\text{recv}_{bC}, \text{ECU}_1^{bC}, \text{ecukey}, \text{ecucnt}, \text{prevcnt}, \text{bus}, \text{mackey}, \text{msgid}, \text{msg}, \text{cnt}) \in \text{alph}(\omega)$ and consider an arbitrary $x \in \lambda^{-1}_{\text{ECU}_1^{bC}}(\lambda_{\text{ECU}_1^{bC}}(\omega)) \cap W_{bCnt}$ (which by definition of $\lambda_{\text{ECU}_1^{bC}}$ and $B_{bCnt} \subseteq W_{bCnt}$ contains b). Since $\text{ECU}_1^{bC} \in \mathcal{ECU}_{bC}$, by Prop.B2 it follows mackey = key = ecukey. Further, by Prop.B3 there is an action $a_1 := (\text{send}_{bC}, \text{ecu}_1, \text{ecukey}_1, \text{ecucnt}_1, \text{prevcnt}_1, \text{bus}, \text{mackey}, \text{msgid}, \text{msg}, \text{cnt}) \in \text{alph}(x)$ before the receive action b by ECU_1^{bC} containing the same message ID, message and counter value and with $\text{ecukey}_1 = \text{mackey}$. Again by Prop.B2 it follows $\text{ecu}_1 \in \mathcal{ECU}_{bC}$ which proves that the message received in b has authentically for ECU_1^{bC} been generated and sent by a member of $\mathcal{ECU}_{bC}$.

Further, by Prop.B1 the receive action b by ECU_1^{bC} is preceded by a send action $\sigma(b) = (\text{send}_{bC}, \text{ecu}\prime, \text{ecukey}\prime, \text{ecucnt}\prime, \text{prevcnt}\prime, \text{bus}, \text{mackey}, \text{msgid}, \text{msg}, \text{cnt})$ triggering b.

Let ν maximal in $\Phi(\sigma(b), W_{bCnt})$ with $b \in \text{alph}(\nu)$. Since $\widehat{\kappa}_{\text{aname}}(b) = \text{recv}_{bC}$, Prop.B12 implies $\widehat{\kappa}_{\text{errorFrame}}(b) = \text{no}$ and thus $\widehat{\kappa}_{\text{errorFrame}}(c) = \text{no}$ for all $c \in \text{alph}(\nu)$ with $\widehat{\kappa}_{\text{aname}}(c) \in \{\text{read}_{bC}, \text{recv}_{bC}\}$. By Prop.B15 one of the actions read_{bC}, recv_{bC} in ν is performed by an ECU owning the correct counter, i.e. there exists $c^* \in \text{alph}(\nu)$ with $\widehat{\kappa}_{\text{aname}}(c^*) \in \{\text{read}_{bC}, \text{recv}_{bC}\}$, $\text{ECU}^*(\sigma(b)) := \widehat{\kappa}_{\text{ecu}}(c^*) \in \mathcal{ECU}_{bC}$, $\widehat{\kappa}_{\text{errorFrame}}(c^*) = \text{no}$ and $\widehat{\kappa}_{\text{prevcnt}}(c^*) = \text{corCnt}_{bC}(c^*, x) + 1$. Lemma 3 implies $\widehat{\kappa}_{\text{ecucnt}}(c^*) = \text{corCnt}_{bC}(c^*, x)$. Recall that Prop.B1 and Prop.B3 imply that the values of the parameters msg, msgid and cnt in $a_1, \sigma(b), b$ and c^* are identical.

(1) Assume $\widehat{\kappa}_{\text{msgid}}(c^*) = \text{fmsg}$ and assume further that when sending the message, ecu_1 is synchronized, i.e. includes the correct counter as the message's counter value. By Prop.B9 it follows $\widehat{\kappa}_{\text{cnt}}(a_1) = \widehat{\kappa}_{\text{prevcnt}}(a_1) - 1 = \text{corCnt}_{bC}(a_1, x)$. Since all read_{bC} and recv_{bC} actions in ν have errorFrame = no, Prop.B11 implies $\widehat{\kappa}_{\text{cnt}}(c^*) = \widehat{\kappa}_{\text{ecucnt}}(c^*)$, hence $\widehat{\kappa}_{\text{cnt}}(c^*) = \text{corCnt}_{bC}(c^*, x)$. It follows $\widehat{\kappa}_{\text{cnt}}(\sigma(b)) = \widehat{\kappa}_{\text{cnt}}(a_1) = \widehat{\kappa}_{\text{cnt}}(c^*) = \text{corCnt}_{bC}(c^*, x)$. Now if $a_1 \neq \sigma(b)$, the number of send_{bC} actions until a_1 is smaller than the number of send_{bC} actions until $\sigma(b)$. Since by Lemma 2 the correct counter minus any number of send_{bC} actions is always bigger or equal to 0, it follows that the correct counter for $\sigma(b)$ is different to the one for a_1. More specifically, it is smaller, i.e. $\text{corCnt}_{bC}(\sigma(b), x) \leq \text{corCnt}_{bC}(a_1, x) - 1$. Further, again by Lemma 2, the number of send actions having occurred until $\sigma(b)$ is equal to those having occurred until a read_{bC} or recv_{bC} action induced by $\sigma(b)$, i.e. $\text{corCnt}_{bC}(\sigma(b), x) = \text{corCnt}_{bC}(c^*, x)$. This implies $\widehat{\kappa}_{\text{cnt}}(a_1) = \widehat{\kappa}_{\text{cnt}}(\sigma(b)) = \widehat{\kappa}_{\text{cnt}}(c^*) = \text{corCnt}_{bC}(c^*, x) = \text{corCnt}_{bC}(\sigma(b), x) \leq \text{corCnt}_{bC}(a_1, x) - 1$. This constitutes a contradiction to our assumption that ecu_1 is synchronized in a_1 and sends $\widehat{\kappa}_{\text{cnt}}(a_1) = \text{corCnt}_{bC}(a_1, x)$. Thus $a_1 = \sigma(b)$.

(2) Assume $\widehat{\kappa}_{\text{msgid}}(c^*) = \text{sync}$ and ecu_1 is synchronized which by Prop.B7 implies that it sends the correct counter as the message's payload, i.e. $\widehat{\kappa}_{\text{msg}}(a_1) = \text{corCnt}_{bC}(a_1, x)$. Further, c^* being a read_{bC} or recv_{bC} action with $\widehat{\kappa}_{\text{errorFrame}}(c^*) = \text{no}$, Prop.B8 implies $\widehat{\kappa}_{\text{ecucnt}}(c^*) = \widehat{\kappa}_{\text{msg}}(c^*) \leq \widehat{\kappa}_{\text{prevcnt}}(c^*) - 1$. Hence it follows $\text{corCnt}_{bC}(c^*, x) = \widehat{\kappa}_{\text{ecucnt}}(c^*) = \widehat{\kappa}_{\text{msg}}(c^*) \leq$

$\widehat{\kappa}_{\text{prevcnt}}(c^*) - 1 = \text{corCnt}_{bC}(c^*, x)$ and thus $\widehat{\kappa}_{\text{msg}}(c^*) = \text{corCnt}_{bC}(c^*, x)$. Assume $a_1 \neq \sigma(b)$. As above, Lemma 2 implies $\text{corCnt}_{bC}(\sigma(b), x) \leq \text{corCnt}_{bC}(a_1, x) - 1$ and $\text{corCnt}_{bC}(\sigma(b), x) = \text{corCnt}_{bC}(c^*, x)$. Since by Prop.B1 $\widehat{\kappa}_{\text{msg}}(a_1) = \widehat{\kappa}_{\text{msg}}(\sigma(b)) = \widehat{\kappa}_{\text{msg}}(c^*)$, it follows $\text{corCnt}_{bC}(a_1, x) = \widehat{\kappa}_{\text{msg}}(a_1) = \widehat{\kappa}_{\text{msg}}(c^*) = \text{corCnt}_{bC}(c^*, x) = \text{corCnt}_{bC}(\sigma(b), x) \leq \text{corCnt}_{bC}(a_1, x) - 1$. This again constitutes a contradiction, thus it follows $a_1 = \sigma(b)$.

(3) Assume ecu_1 is not synchronized in a_1. By Prop.B16 it has performed an action loseCnt_{bC} before a_1, denoted by a_2. If $a_2 \neq \text{pre}_1(x)$, by Prop.B17, $\widehat{\kappa}_{\text{ecucnt}}(a_2)$ is set to a value bigger than the action's correct counter value: $\widehat{\kappa}_{\text{ecucnt}}(a_2) > \text{corCnt}_{bC}(a_2, x)$. If $a_2 = \text{pre}_1(x)$, Prop.B17 implies $\widehat{\kappa}_{\text{ecucnt}}(a_2) = bCnt_{\max} + 1 > \text{corCnt}_{bC}(a_2, x)$ as well. Assume for simplicity that ecu_1 's next action after a_2 is a_1. Then $\widehat{\kappa}_{\text{prevcnt}}(a_1) = \widehat{\kappa}_{\text{ecucnt}}(a_2) > \text{corCnt}_{bC}(a_2, x)$. Let $\widehat{\kappa}_{\text{msgid}}(a_1) = \text{fmsg}$. Then by Prop.B9, $\widehat{\kappa}_{\text{cnt}}(a_1) = \widehat{\kappa}_{\text{prevcnt}}(a_1) - 1 = \widehat{\kappa}_{\text{ecucnt}}(a_2) - 1 > \text{corCnt}_{bC}(a_2, x) - 1$. Now a_1 is the first send_{bC} action after a_2 since a send_{bC} action in between would imply another action by ecu_1 (Prop.B14). Lemma 2 implies $\text{corCnt}_{bC}(a_2, x) - 1 = \text{corCnt}_{bC}(a_1, x)$. Hence ecu_1 not being synchronized results into $\widehat{\kappa}_{\text{cnt}}(a_1) > \text{corCnt}_{bC}(a_1, x)$. If a_2 is not ecu_1 's last action before a_1 we can argue analogously only with a longer sequence of actions and counters in between a_2 and a_1 to consider, being decreased step by step. By Prop.B1 $\widehat{\kappa}_{\text{cnt}}(a_1) = \widehat{\kappa}_{\text{cnt}}(\sigma(b)) = \widehat{\kappa}_{\text{cnt}}(c^*)$ always holds. Hence $\text{corCnt}_{bC}(a_1, x) < \widehat{\kappa}_{\text{cnt}}(c^*) = \widehat{\kappa}_{\text{cnt}}(a_1)$. By Lemma 2 and the definition of corCnt_{bC}, $\text{corCnt}_{bC}(\sigma(b), x) = \text{corCnt}_{bC}(c^*, x)$, and since ECU^* owns the correct counter in c^*, by Prop.B15 it follows $\widehat{\kappa}_{\text{prevcnt}}(c^*) = \text{corCnt}_{bC}(c^*, x) + 1$ and Lemma 3 implies $\widehat{\kappa}_{\text{ecucnt}}(c^*) = \text{corCnt}_{bC}(c^*, x)$. Together these statements imply $\text{corCnt}_{bC}(\sigma(b), x) = \widehat{\kappa}_{\text{ecucnt}}(c^*)$. Further, Lemma 2 implies $\text{corCnt}_{bC}(a_1, x) \geq \text{corCnt}_{bC}(\sigma(b), x)$, no matter whether or not a_1 and $\sigma(b)$ are identical. Hence $\widehat{\kappa}_{\text{ecucnt}}(c^*) = \text{corCnt}_{bC}(\sigma(b), x) \leq \text{corCnt}_{bC}(a_1, x)$ and therefore $\widehat{\kappa}_{\text{ecucnt}}(c^*) \leq \text{corCnt}_{bC}(a_1, x) < \widehat{\kappa}_{\text{cnt}}(c^*)$. Prop.B11 implies $\widehat{\kappa}_{\text{errorFrame}}(c^*) = \text{yes}$, and by Prop.B12 it follows $\widehat{\kappa}_{\text{errorFrame}}(b) = \text{yes}$ and therefore $\widehat{\kappa}_{\text{aname}}(b) \neq \text{recv}_{bC}$, a contradiction to the assumption we started the proof with. Hence $\text{corCnt}_{bC}(a_1, x) = \widehat{\kappa}_{\text{cnt}}(a_1)$ in the case of $\widehat{\kappa}_{\text{msgid}}(a_1) = \text{fmsg}$.

If $\widehat{\kappa}_{\text{msgid}}(a_1) = \text{sync}$, we can decude $\widehat{\kappa}_{\text{errorFrame}}(b) = \text{yes}$ and thus the same contradiction by exchanging the parameter cnt by msg and applying Prop.B6 instead of Prop.B9 to deduce $\widehat{\kappa}_{\text{msg}}(a_1) > \text{corCnt}_{bC}(a_1, x)$. Further, Lemma 2 implies $\widehat{\kappa}_{\text{msg}}(a_1) = \widehat{\kappa}_{\text{msg}}(\sigma(b)) = \widehat{\kappa}_{\text{msg}}(c^*)$ and it follows $\widehat{\kappa}_{\text{msg}}(c^*) > \text{corCnt}_{bC}(a_1, x)$. With the same arguments as above, this implies $\widehat{\kappa}_{\text{msg}}(c^*) > \widehat{\kappa}_{\text{ecucnt}}(c^*)$. Prop.B8 implies $\widehat{\kappa}_{\text{errorFrame}}(c^*) = \text{yes}$ and again by by Prop.B12 it follows $\widehat{\kappa}_{\text{errorFrame}}(b) = \text{yes}$ and therefore $\widehat{\kappa}_{\text{aname}}(b) \neq \text{recv}_{bC}$. So again the assumption of ecu_1 not being synchronized leads to a contradiction. This concludes our proof.

The above, proof shows that our approach indeed satisfies data origin authenticity as well as immediacy and nonrepeatability. In contrast, the generic counter system violates the latter ones. In the next section, we will discuss the security related differences in more detail. We will then introduce our proof of concept implementation showing its practicability and design decisions that substantiate our formal proof. □

7. Evaluation

In this section, we will evaluate both the security and the practicability of our bus counter approach. More specifically, in the next section we will discuss the formal proof results concerning the satisfaction of the security requirements immediacy and non-repeatability by the generic and the BusCnt system, respectively, and highlight the differences. In Section 7.2 we will then demonstrate the feasibility of our BusCnt approach based on a practical implementation and discuss design decisions.

7.1. Security Aspects. One fundamental difference between our approach and the generic counter-based approach is that in the BusCnt system the pulse generator is an integral component of the system itself: The very writing onto the bus causes a change of the local bus counter values of all ECUs connected to it as they inevitably read (part of) the message (even if not accepting it) and decrement their counters. By this read action, the message's counter and thus its MAC is invalidated. Hence, any subsequent message written onto the bus must use a smaller counter in order to be accepted. This prohibits message delay and replication.

Another important aspect is the assignment of computations to the controller that in traditional CAN communication systems is processed by the application layer. This concerns in particular MAC calculation and verification and checks regarding the size of the message's counter. Performing these checks on controller level enables to use the error frame mechanism of CAN in case of a failed check which in return results in invalidation of the respective message and prevention of its acceptance by any of the ECUs connected to the same bus.

These two aspects together allowed to formally prove that the BusCnt system satisfies both immediacy and nonrepeatability, provided at least one of the ECUs owns the correct counter. The inevitable decrease of the counter value with every new message prohibits message delay, and the checks performed by the controllers allow immediate invalidation of any manipulated message.

One of the core assumptions of our proof is the correctness of at least one counter value in the bus network. It is based on two observations: First, ECUs are designed to be safe and thus hardware or software failures occur significantly less often compared to regular PC hardware. Second, an incorrect counter value may be the result of the

shutdown process, as an ECU may not be able to store the counter value in time to persistent storage. However, the likelihood for this failure to occur is reduced in our approach since only a single value per bus needs to be stored. In contrast, traditional counter-based approaches use various counters for different types of messages. Moreover, a bus has a large number of ECUs connected to it and only one needs to store the correct counter value at the end of a ride. Therefore, the probability that our assumption does not hold is insignificant.

On the other hand, an ECU not being synchronized, i.e. not owning the correct counter, may in principle cause a safety problem: If it sends a functional message, MAC verification by ECUs owning the correct counter will fail and result into an error frame. If the ECU itself receives a functional message containing the correct counter, its own MAC verification will fail and cause an error frame as well. Too many error frame events will cause the ECU to change into an inactive state. This safety problem can be minimized by adequate synchronization approaches as discussed in Section 7.2.2 below.

In contrast to the BusCnt system, our proof of the generic counter-based approach indicates that it exhibits several weaknesses. First, it cannot ensure immediacy and non-repeatability (neither of synchronization nor of functional messages) in case an ECU loses its counter. Once being active again the ECU will accept any replayed message whose counter is still in the required range (i.e. bigger than its own counter). This violates immediacy, assuming that only the period between writing a message onto the bus and reading it does not exceed a specified limit. It also violates non-repeatability since the attack is possible even if the replayed message has already been accepted before.

The counter synchronization mechanism of GenCnt does not prohibit this attack as synchronization messages themselves are susceptible to delay attacks, i.e. recorded and then invalidated by an attacker by destroying their CRC or by interrupting the message with an error frame. These messages will then not be accepted by the ECUs with the consequence that an unsynchronized ECU cannot be synchronized. It will therefore accept delayed synchronization and functional messages at any later point in time the attacker chooses for a replay, thus violating immediacy. Hence, the synchronization mechanism of the generic counter-based system is no guarantee for ensuring immediacy and non-repeatability.

Even if no counter loss occurs, violation of immediacy by a delay of messages cannot be prohibited. This is due to the fact that the counter value stored by the intended recipients of a message does not change as long as all messages relevant for the respective counter are invalidated by the attacker and thus not accepted. Consequently, the intended recipients will accept any relayed message since it still contains a counter being valid from their point of view.

One question that comes to mind is whether it would be sufficient to equip the Fresh Value Manager FvM of the GenCnt system with the ability to perform all security checks by the controller in order to avoid the above-described attacks. However, it turns out that this measure alone is not enough. First, the FvM behavior would need to be changed as it must increment its own local counter value with every sent message, independently of whether or not the message is accepted. In other words, the determination of what is the correct counter for a message would need to be adapted to the one used for the BusCnt system. Otherwise, the FvM would not be able to detect the repetition of a message as it would still consider the old counter to be correct. Since the sender of a message that has caused an error frame normally simply resets its counter and resends the message, the FvM would additionally need to adjust the sender's counter. Secondly, the system would need to be changed to using one single counter per bus since otherwise the FvM is not able to assign an error frame to the counter used in a message that has been interrupted before writing the message ID to the bus. All these changes result in a system that is in some of its main aspects equivalent to the BusCnt system.

There are some assumptions in our proof regarding the satisfaction of immediacy and non-repeatability by the BusCnt system that need to be substantiated by specific design decisions. This concerns for example the size of the counter that must prohibit overflow. In the BusCnt system, an attacker can accelerate the pulse generator by inserting messages onto the bus. Independently of being accepted, they will cause the connected ECUs to decrement their counters faster than they normally would. However, in Section 7.2.2 below, we discuss the counter length necessary to avoid counter overflow even in the presence of such an attack and show how this length can be implemented in praxis. Other assumptions concern storage errors that we assume never to result in a counter being smaller than the correct one as this could lead to counter overflow as well. This is an issue for all counter-based approaches, adequate measures for detection of such incidents is out of the scope of this paper.

A final security aspect concerns the truncation of MACs which in principle enables an attacker to construct its own message and to determine the corresponding truncated MAC by brute force. This holds in particular for synchronization messages. So if an attacker was able to construct and insert a synchronization message containing e.g. the counter 0...0 with a correct MAC, it could take over the whole bus communication. This is an issue for all approaches using MAC truncation (e.g. for AUTOSARs SecOC). As a counter-measure, we have chosen a specific number of failed synchronization messages as indicator of a brute force attack which we deem small enough to recognize such attacks and on the other hand big enough to not cause unnecessary dysfunction of the bus. See Section 7.2.2 for more details on this aspect.

7.2. Practical Aspects. To evaluate the practical aspects of our work we implemented a proof of concept of BusCount to demonstrate the general feasibility of the mechanisms. Moreover, we want to show the suggested approach can be implemented at a low cost. We first introduce our development setup and describe the design decisions before presenting the evaluation we performed bsaed on this implementation.

7.2.1. Setup. For the implementation of our security enhanced CAN controller, we chose a low-cost FPGA (ICE40HX8K-B-EVN) with only 7680 logic cells and a maximum frequency of 12 MHz. The FPGA is connected with a CAN transceiver (MCP2561) that converts logical CAN messages into physical signals for a CAN bus.

We used an open-source implementation1 as a basis for our CAN controller. We extended the controller with an SLCAN2 protocol to communicate with a connected ECU. The ECU is simulated by a Raspberry Pi 2B running a default Linux SocketCAN3. The software does not need to be modified besides the fact that the payload is reduced by the length of the MAC. A second FPGA with the same setup was introduced to perform MAC verifications and synchronization tests. The correctness of the CAN implementation and the compatibility with regular CAN bus devices was evaluated with a remaining bus simulation using a Vector VN5610 in combination with CANoe v9. The hardware setup of our proof of concept (see Figure 5) contains:

7.2.2. Design Decisions. We decided to use a truncated MAC value with a size of 24 bit to be compatible with AUTOSAR SecOC. Since CAN has a very limited message size per package we decided to use the counter implicitly. This requires more explicit synchronizations yet does not disrupt the system's functionality due to the fast synchronization mechanism of our protocol. Corresponding to the 24 bit MAC the remaining payload of a CAN message is 40 bits. The counter transferred with the described synchronization mechanism would then also be restricted to these 40 bits. A CAN bus can transmit up to 17,543 messages per second [36], thus a 40 bit counter suffices for about 725.4 days $(2^{40}/17,543 \cdot 60 \cdot 60 \cdot 24)$ of non-stop communication before an overflow occurs. An attacker may even reduce the duration by starting CAN messages and stopping them immediately with an error frame. This increases the number of messages sent by an attacker (15 bits per message) compared to a regular sender (minimum 44 bits per message) to about 51,459 messages per second $(17,543 \cdot 44/15)$. We consider a counter value that could overflow after only 247.3 days $(2^{40}/51,459 \cdot 60 \cdot 60 \cdot 24)$ not sufficient in an attack scenario, thus we increased the counter by additional 18 bits which can be transmitted using the extended message ID of the CAN specification for synchronization messages. The 58 bit counter is sufficient for about 82,884.75 years $(2^{58}/51,459 \cdot 60 \cdot 60 \cdot 24 \cdot 365)$. In order to counter brute force attacks and to increase the security of the truncated MAC we suggest renewing the key regularly by deriving it from the current counter value.

Furthermore, we needed to make sure the synchronization cannot be attacked by a brute force attack. An attacker may try to forge a synchronization message setting the counter to 0 which would lead to an overflow or would establish the number 0 as the counter value of all subsequent messages. Since we suggest to transfer only a 24 bit MAC, attacks cannot be prevented by the key size. For this reason we count the number of failed synchronizations. If more than 16 failed synchronizations during a car ride or 128 failed synchronizations in total have been detected, the ECUs need to consider the bus no longer trustworthy and must enable the driver to safely stop the car. The 128 synchronizations give an attacker a 0.000977% $\left(\binom{128}{1}\binom{2^{24}}{127}/\binom{2^{24}}{128}\right)$ chance to forge a synchronization message successfully. A recovery process for a car network is out of the scope of this paper. Compared to the generic counter-based approach, our synchronization solution has the advantage that it is independent of functional disruptions regarding a central entity (fresh value master): Any ECU can initialize synchronization, and all ECUs join in, sending their respective synchronization messages simultaneously. This mechanism allows the synchronization of all ECUs connected to one bus during the transmission time of only one message.

The bus counter-based security protocol does not change the CAN frame and thus it can work with default CAN controllers in one network. The adaptation to CAN FD and CAN XL is also possible. Further, it is compatible with SecOC with respect to the message structure.

The introduction of error frames in conjunction with security checks potentially changes the safety considerations of ISO26262 regarding CAN bus communication. A CAN controller sending a message which results in an error frame increments its error counter by 8 while receiving controllers increment their error counter by 1. Successful message transfer reduces the counter by 1. If an error counter exceeds the value 128, the respective controller changes into `bus-off` mode and is not able to send or receive data anymore and a hardware reset is necessary. However, there are only two cases that our security mechanisms could cause a `bus-off` state. In the first case, the attacker is the sender of unauthorized messages by delay, replay, or forging of messages. In this scenario, the attacker performs a denial of service attack which she could also perform in every other secure or non-secure CAN BUS by inserting error frames. The second scenario is the startup of a vehicle where every CAN controller is out of synchronization. In the worst case, ECUs might join the network one by one, each of them sending an error frame caused by a failed check based on its wrong counter value, and then initiate a synchronization. To prevent this we suggest that after waking up and before starting to send error frames, a CAN controller first initiates a synchronization in case of an invalid message. Thus, the ECU is in sync as soon as it joins the network.

7.2.3. Performance Evaluation. The CAN controller implementation we adapted to work on the ICE40HX8K consumed 4,483 of the 7,680 logic cells. The remaining logic cells need to be sufficient for processing a MAC algorithm, the synchronization, and the counter mechanism. Moreover, the MAC algorithm needs to be fast enough to calculate the MAC during the transmission of the truncated MAC value $(24 \cdot 4 = c)$. The time consumed by the sending process is

CAN controller	ICE40HX8K-B-EVN
CAN transceiver	MCP2561
ECU	Raspberry Pi 2B
Remaining bus simulation	Vector VN5610

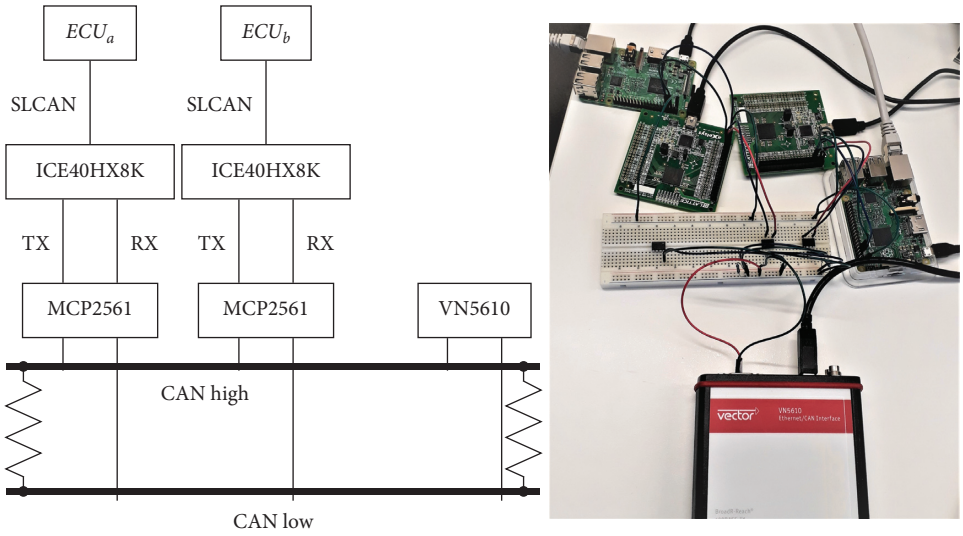

Figure 5: Evaluation setup for BusCount.

longer since much more bits are transmitted prior to the truncated MAC value. Finally, the MAC algorithm needs to calculate the MAC over at most 131 bits (58 bit counter value + 29 ID + 4 bit data length + 40 bit data) for classic CAN.

The remaining 3,197 cells are not sufficient to implement a regular cipher, like AES or HMAC with SHA2, with low latency together with the counter and synchronization mechanism. For this reason, we evaluated several lightweight ciphers regarding their number of rounds, state size, and table size. Based on these results we implemented our CAN controller in combination with the two CBC-MAC algorithms Present80 [37] and Prince [38] and the HMAC algorithm SipHash [39]. All three have a block size of 64 bit, so two blocks need to be processed at most. We evaluated the number of cycles each algorithm needs for this task. Table 6 shows the results of our evaluation. The number of cycles as well as the number of logical cells needed for the CAN controller including the different ciphers.

The smallest (in terms of cells needed) but also the slowest algorithm was Present80. Our implementation of the controller with Present80 needed 5,599 logic cells and 68 cycles for two blocks. Prince only needed 30 cycles while increasing the number of logic cells needed to 5,947. SipHash was only slightly larger with 6,024 logic cells, but needs only 13 cycles to compute a MAC over two blocks. Since the security of Present80 is lower and the number of blocks increases drastically regarding CAN FD (10 blocks for 579 bits of authentic data) and CAN XL (258 blocks for at most 16,466 bits of authentic data), we recommend SipHash for a fast and size efficient MAC algorithm in our approach. Additionally, cryptanalysises [40,41] of SipHash did not reveal problems with this cryptographic primitive.

Table 6: Evaluation of ciphers for secure CAN controller.

Algorithm	Cycles	Logic cells (total)
Plain CAN controller	-	4,483
SipHash [39]	13	6,024
Prince [38]	30	5,947
Present80 [37]	68	5,599

8. Conclusions and Future Work

In this paper, we have presented a detailed discussion and formal evaluation of our hardware-based approach BusCount for the security protection of automotive CAN networks. We further opposed this to the characteristics of counter-based approaches currently being used.

The fundamental difference between currently considered approaches and ours is that in ours the pulse generator is an integral component of the system itself: The very writing onto a bus causes a change of the counter values of all ECUs connected to it as they inevitably read the message (even if not accepting it) and decrement their counters. Since the number of messages sent on the bus cannot be manipulated, the correct counter value cannot be manipulated as well. By this read action, the message's counter and thus its MAC is invalidated. Another important aspect is the assignment of MAC calculation and verification and checks regarding the size of the message's counter to the controller. This enables to use the error frame mechanism of CAN communication in case of a failed check which results in invalidation of all messages whose MAC is not based on the correct counter value. These messages will then not be accepted by any of the ECUs. Software-based approaches do not have this possibility since they verify the MAC on

application-level after the controller completely received the message.

We formally proved that our bus counter approach satisfies data origin authenticity as well as immediacy and non-repeatability (also denoted by message freshness), both during regular operation and in case an ECU loses its counter. Our proof is based on the assumption that simultaneous loss of counter values does not occur, i.e. that there is always at least one ECU per bus owning the correct value. This assumption seems appropriate, given the low possibility of it being violated. It enables our synchronization mechanism to take advantage of the physical characteristics of a CAN bus and ensures that always the correct counter value is sent.

On the other hand, current counter-based systems cannot assure message freshness if an ECU loses its counter. Immediacy is even violated in case an ECU does not lose its counter: An attacker can invalidate all messages relevant for a specific counter and insert them again at a later point in time without the ECU being able to notice this manipulation. This is due to the fact that the ECU's local counter value only changes if it actually accepts a message.

Compared to other approaches, our bus counter mechanism offers several practical advantages: It avoids the necessity to include (parts of) the counter in the messages which saves bandwidth, and it requires only one counter per bus to be stored instead of one counter per message ID favored by currently discussed approaches. This reduces both storage capacities and the risk of ECUs being unsynchronized.

The synchronization solution of BusCount has the advantage that it is independent of functional disruptions regarding a central fresh value master that is being used by other counter-based approaches: Any ECU can initialize synchronization, and all ECUs join in, sending their respective synchronization messages simultaneously. This mechanism allows the synchronization of all ECUs connected to the same bus during the transmission time of only one message.

It must be noted that our approach cannot be realized with currently available ECUs. On the other hand, it does not change the CAN frame and can thus work with default CAN controllers in one network. It can also be adapted to CAN FD and CAN XL and is compatible with SecOC with respect to the message structure. Moreover, our proof of concept implementation described in Section 7.2 shows that our approach is not only theoretically interesting, but is functionally working in a CAN network. However, an implementation needs to respect a couple of considerations. One of them is that our approach allows pulse acceleration, hence the counter must be sufficiently long in order to prohibit counter overflow during the lifetime of a car. We consider 58 bits as suggested in Section 7.2.2 adequate. Further, a concrete synchronization mechanism must for example prohibit brute force attacks on truncated MACs of synchronization messages and must ensure a synchronized network as soon as possible after the startup process.

In the future, we plan to extend our attack model and address an adversary that can manipulate devices connected to the bus. One possible countermeasure that at least reduces the severity of this attack is to assign specific roles to different ECUs and to enable ECUs to authentically identify the sender of messages. Hence, we plan to investigate how our approach can be extended by sender authentication mechanisms. Further, we will explore whether and how techniques to protect the devices' integrity can be integrated in order to prohibit manipulation of genuine ECUs.

Data Availability

No data were used to support this study.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This research work has been partly funded by the German Federal Ministry of Education and Research (BMBF) and the Hessen State Ministry for Higher Education, Research and the Arts within their joint support of the National Research Center for Applied Cybersecurity ATHENE and by the BMBF project VITAF (ID 16KIS0835).

References

[1] Road Vehicles - Controller Area Network (CAN), *Standard*, International Organization for Standardization, Geneva, Switcherland, 2015.

[2] K. Koscher, A. Czeskis, F. Roesner et al., "Experimental security analysis of a modern automobile," in *Proceedings of the 31st IEEE Symposium on Security and Privacy*, pp. 447–462, IEEE Computer Society, Oakland, CA, USA, May 2010.

[3] H. Schweppe, "Deliverable D3.3: secure on-board protocols specifcation," Technical report, EVITA, Panaji, India, 2011.

[4] Autosar, "Specification of secure Onboard communication," 2018, https://www.autosar.org/standards/classic-platform/classic-platform-440.

[5] S. D. Gürgens and D. Zelle, "A hardware based solution for freshness of secure onboard communication in vehicles," in *Computers & Security*, Sokratis K. Katsikas et al., Ed., in *Proceedings of the Computer Security - ESORICS 2018 International Workshops, CyberICPS 2018 and SECPRE 2018*, vol. 11387, pp. 53–68, Springer, Barcelona, Spain, September 2018.

[6] C. Miller and C. Valasek, "A survey of remote automotive attack surfaces," in *Proceedings of the Black Hat USA*, Lag Vegas, NJ, USA, August 2014.

[7] D. Pohler, A. Tim, K. Chsristopher, H. Martin, L. Johannes, and P. Ulrich, "Real driving NOx emissions of European trucks and detection of manipulated emission systems," in *Proceedings of the EGU General Assembly Conference Abstracts*, vol. 19, p. 13991, Vienna, Australia, April 2017.

[8] I. D. Foster, A. Prudhomme, K. Koscher, and S. Savage, "Fast and vulnerable: a story of telematic failures," Edited by Aurelien Francillon and Thomas Ptacek. USENIX Association, Ed., in *Proceedings of the 9th USENIX Workshop on Offensive Technologies, WOOT 215*, vol. 2015, Washington, DC, USA, August, 2015.

[9] G. Bella, P. Biondi, G. Costantino, and I. Matteucci, "TOU-CAN: a proTocol tO secUre Controller Area Network," in

Proceedings of the ACM Workshop on Automotive Cybersecurity, Ziming Zhao, Qi Alfred Chen, and Gail-Joon Ahn, Ed., vol. 2019, pp. 3–8, AutoSec@CODASPY, Richardson, TX, USA, March 2019.

[10] H. Ahmed and F. Ha, "Lcap-a lightweight can authentication protocol for securing in-vehicle networks"" in *Proceedings of the escar Embedded Security in Cars Conference*, vol. 10, isits AG International School of IT Security, Berlin, Germany, November 2012.

[11] S. Woo, H. J. Jo, I. S. Kim, and D. H. Lee, "A practical security architecture for in-vehicle CAN-FD," *IEEE Transactions on Intelligent Transportation Systems*, vol. 17, pp. 2248–2261, 2016.

[12] S. Nürnberger and C. Rossow, "vatiCAN - vetted, authenticated CAN bus," in *Proceedings of the Cryptographic Hardware and Embedded Systems - CHES 2016 - 18th International Conference*, Benedikt Gierlichs and Axel Y. Poschmann, Ed., vol. 9813, pp. 106–124, Springer, Santa Barbara, CA, USA, August, 2016.

[13] J. Van Bulck, J. T. Mühlberg, and F. Piessens, "VulCAN," in *Proceedings of the 33rd Annual Computer Security Applications Conference*, pp. 225–237, ACM, Orlando, FL, USA, December 2017.

[14] O. Hartkopp, C. Reuber, and R. Schilling, "MaCAN-message authenticated CAN," in *Proceedings of the escar Embedded Security in Cars Conference*, vol. 10, isits AG International School of IT Security, Berlin, Germany, November 2012.

[15] A. Bruni, M. Sojka, F. Nielson, and H. Riis Nielson, "Formal security analysis of the MaCAN protocol," in *Proceedings of the Integrated Formal Methods - 11th International Conference, IFM 2014*, Elvira Albert and Emil Sekerinski, Ed., vol. 8739, pp. 241–255, , Springer, Bertinoro, Italy, September, 2014.

[16] R. Kurachi, Y. Matsubara, H. Takada, H. Ueda, and S. Horihata, "CaCAN-centralized authentication system in CAN (controller area network)," in *Proceedings of the escar Embedded Security in Cars Conference*, vol. 14, isits AG International School of IT Security, Hamburg, Germany, November 2014.

[17] A. Groll and C. Ruland, "Secure and authentic communication on existing in-vehicle networks," in *Proceedings of the 2009 IEEE Intelligent Vehicles Symposium*, pp. 1093–1097, IEEE, Xi'an, China, June 2009.

[18] C.-W. Lin, L. Alberto, and S. Vincentelli, "Cyber-security for the controller area network (CAN) communication protocol," in *Proceedings of the 2012 ASE International Conference on Cyber Security*, pp. 1–7, IEEE Computer Society, Alexandria, VA, USA, December, 2012.

[19] A.-I. Radu, D. Flavio, and G. LeiA, "A lightweight Authentication protocol for CAN," in *Proceedings of the Computer Security - ESORICS 2016 - 21st European Symposium on Research in Computer Security*, Ioannis G. Askoxylakis et al., Ed., vol. 9879, pp. 283–300, Springer, Heraklion, Greece, September, 2016.

[20] Q. Wang and S. Sawhney, "VeCure: a practical security framework to protect the CAN bus of vehicles""vol. 2014, pp. 13–18, in *Proceedings of the 4th International Conference on the Internet of Things, IOT 2014*, vol. 2014, IEEE, Cambridge, MA, USA, October, 2014.

[21] T. Ziermann, S. Wildermann, and J. Teich, "CAN+: a new backward compatible Controller Area Network (CAN) protocol with up to 16x higher data rates," in *Proceedings of the Design, Automation and Test in Europe, DATE 2009*, L. Benini, Ed., vol. 2009, pp. 1088–1093, , IEEE, Nice, France, April 2009.

[22] A. Van Herrewege, S. Dave, and I. Verbauwhede, "CANAuth-a simple, backward compatible broadcast authentication protocol for CAN bus," in *Proceedings of the ECRYPT Workshop on Lightweight Cryptography*, vol. 2011, Louvain-la-Neuve, Belgium, November 2011.

[23] B. Groza, S. Murvay, A. V. Herrewege, and I. Verbauwhede, "LiBrA-CAN: a lightweight broadcast authentication protocol for controller area networks," in *Proceedings of the Cryptology and Network Security, 11th International Conference, CANS 2012*, Josef Pieprzyk, Ahmad-Reza Sadeghi, and Mark Manulis, Ed., vol. 7712, pp. 185–200, Springer, Darmstadt, Germany, December, 2012.

[24] A. S. Siddiqui, J. Plusquellic, Y. Gui, and F. Saqib, "Secure communication over CANBus," in *Proceedings of the 2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS)*, pp. 1264–1267, Boston, MA, USA, August 2017.

[25] H. Ueda, R. Kurachi, H. Takada, T. Mizuthani, M. Inoue, and S. Horihat, "Security authentication system for in-vehicle network," *SEI Technical Review*, vol. 81, pp. 5–9, 2015.

[26] B. Groza, L. Popa, and P.-S. Murvay, "Highly efficient authentication for CAN by identifier reallocation with ordered CMACs," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 6, pp. 6129–6140, 2020.

[27] Q. Zou, W. Keung Chan, K. C. Gui et al., "The study of secure CAN communication for automotive applications," in *SAE Technical Paper*SAE International, Warrendale, PA, United States, 2017.

[28] A. Muller and T. Lothspeich, "Plug-and-Secure communication for CAN," *CAN Newsletter*, vol. 4, pp. 10–14, 2015.

[29] A. Fuchs, S. Gürgens, and C. Rudolph, "Formal notions of trust and confidentiality- enabling reasoning about system security," *Journal of Information Processing*, vol. 19, pp. 274–291, 2011.

[30] L. Pino, G. Spanoudakis, A. Fuchs, and S. Gürgens, "Generating Secure Service Compositions," in *Proceedings of the International Conference on Cloud Computing and Services Science*, pp. 81–99, Springer, Lisbon, Portugal, May 2015.

[31] A. Fuchs, S. Gurgens, and C. Rudolph, "On the security validation of integrated security solutions," in *IFIP Advances in Information and Communication Technology*, Dimitris Gritzalis and Javier Lopez, Ed., in *Proceedings of the Emerging Challenges for Security, Privacy and Trust - 24th IFIP TC 11 International Information Security Conference, SEC 2009*, vol. 297, Springer, Cyprus, 2009.

[32] B. Hamid, S. Gurgens, and A. Fuchs, "Security patterns modeling and formalization for pattern-based development of secure software systems," *Innovations in Systems and Software Engineering - A NASA Journal ISSN*, vol. 12, pp. 1614–5046, 2015.

[33] S. Eilenberg, "Automata, languages, and machines," *A. Pure and Applied Mathematics*, Academic Press, Cambridge, MA, USA, 1974.

[34] S. Gürgens, P. Ochsenschläger, and C. Rudolph, "Authenticity and provability - a formal framework," in *Infrastructure Security*, George I. Davida,Yair Frankel, and Owen Rees, Ed., in *Proceedings of the Infrastructure Security, International Conference, InfraSec 2002*, vol. 2437, pp. 227–245, Springer, Bristol, UK, October 2002.

[35] R. Grimm and P. Ochsenschläger, "Binding telecooperation - a formal model for electronic commerce," *Computer Networks*, vol. 37, no. 2, pp. 171–193, 2001.

[36] W. Voss, *A Comprehensible Guide to Controller Area Network. Greenfield,Massachusetts*, Copperhill Technologies Corporation, Greenfield, MA, USA, 2008.

[37] A. Bogdanov, L. R. Knudsen, G. Leander et al., "PRESENT: an ultra-lightweight block cipher," in *Proceedings of the Cryptographic Hardware and Embedded Systems - CHES 2007, 9th International Workshop*, Pascal Paillier and Ingrid Verbauwhede, Ed., vol. 4727, pp. 450–466, Springer, Vienna, Austria, September, 2007.

[38] J. Borghoff, A. Canteaut, T. Güneysu et al., "Prince - a low-latency block cipher for pervasive computing applications," in *Proceedings of the Advances in Cryptology - ASIACRYPT 2012 - 18th International Conference on the Theory and Application of Cryptology and Information Security*, Xiaoyun Wang and Kazue Sako, Ed., vol. 7658, pp. 208–225, Springer, Beijing, China, December, 2012.

[39] J.-P. Aumasson and D. J. Bernstein, "SipHash: a fast short-input prf," in *Proceedings of the In: Progress in Cryptology - INDOCRYPT 2012, 13th International Conference on Cryptology in India*, Steven D. Galbraith and Mridul Nandi, Ed., vol. 7668, pp. 489–508, Springer, Kolkata, India, December, 2012.

[40] C. Dobraunig, F. Mendel, and M. Schläffer, "Differential cryptanalysis of SipHash," in *Selected Areas in Cryptography - SAC 2014*, Antoine Joux and Amr Youssef, Ed., Springer International Publishing, New York, NY, USA, pp. 165–182, 2014.

[41] W. Xin, Y. Liu, B. Sun, and C. Li, "Improved cryptanalysis on SipHash," in *Cryptology and Network Security*, Y. Mu, R. H. Deng, and X. Huang, Eds., Springer International Publishing, New York, NY, USA, 2019.