LeiA: A Lightweight Authentication Protocol for CAN

Andreea-Ina Radu and Flavio D. Garcia

School of Computer Science,
University of Birmingham, Birmingham UK
{a.i.radu,f.garcia}@cs.bham.ac.uk

Abstract. Recent research into automotive security has shown that once a single vehicle component is compromised, it is often possible to take full control of the vehicle. This paper proposes LeiA, a lightweight authentication protocol for the Controller Area Network (CAN). This protocol allows critical vehicle Electronic Control Units (ECUs) to authenticate each other providing compartmentalisation and preventing a number of attacks e.g., where a compromised CD player is able to accelerate the vehicle. LeiA is designed to run under the stringent time and bandwidth constraints of automotive applications and is backwards compatible with existing vehicle infrastructure. The protocol is suitable to be implemented using lightweight cryptographic primitives yet providing appropriate security levels by limiting the usage of every key in the system. The security of LeiA is proven under the unforgeability assumption of the MAC scheme under chosen message attacks (uf-cma).

1 Introduction

The automotive industry has recently faced a massive transformation which has enabled serious security threats [8],[4],[15]. The increasing number of (wireless) interfaces available in today's cars exposes it to new attack vectors. Modern Cars have dozens and sometimes even over a hundred of *Electronic Control Units* (ECUs). While more technology is being introduced in modern vehicles, transforming them into smart, connected cars, the underlying security infrastructure has struggled to keep up with the pace of these changes.

The Controller Area Network (CAN), standardised in [13], is the most commonly used serial bus nowadays. Its purpose is to connect the ECUs of a car, and allow them to communicate without a source or destination address. As the in-vehicle network has been traditionally considered a safe, trusted environment, and there were no wireless interfaces, resilience against cyber-attacks has not been of prime concern. Also, the security of ECUs, which provide a significant part of the functionality of a modern vehicle, has been overlooked. The CAN bus is a broadcast network, whereby any message sent can be read by all connected ECUs. By design, it does not provide security features, such as confidentiality (messages are not encrypted, therefore they can be eavesdropped), or authenticity (the source or destination of a message is unknown) [18]. Most attacks presented in the literature could be prevented if authentication was present on the network, or at least their impact would be localised and mitigated.

While transitioning from mostly mechanical systems to complex systems with digital components, manufacturers overlooked the possibility of a cyber-attacker in their designs. The KeeLoq block cipher, used by various car manufacturers in anti-theft mechanisms, was first attacked by Bogdanov in [2]. Later, this attack was improved in [5],[12],[14]. Verdult et al. proposed an attack against the Megamos Crypto [21], [24] and Hitag2 [22] vehicle immobilisers. These attacks allow an adversary to start the vehicle without the car key. Automotive remote keyless entry systems have also been shown to use weak key management and cryptographic primitives, enabling an eavesdropping adversary to clone the car key [7].

Koscher et al. [15] provide an extensive description of attack vectors under the assumption that an attacker has direct access to the vehicle, focusing on the security of the in-vehicle networks. Their research shows it is possible to compromise the radio, instrument panel cluster, HVAC, BCM (which controls door locks, interior and exterior lights, horn, windows, wipers, ignition), as well as safety-critical functionality of the engine and brakes. They launched a generic denial of service attack which disabled communication on the CAN bus and froze the instrument panel cluster. Part of the attacks were then tested on the road, proving their viability in a real-world scenario.

The work of Koscher et al. raises a key issue: *how* would an attacker get access to the vehicle. Under the assumption that prior physical access to the vehicle is deemed as an unrealistic scenario, Checkoway et al. [4] explore the external attack surface of automotive vehicles. They successfully used the entertainment system, radio, Bluetooth interface, Tire Pressure Monitoring System (TPMS) and cellular network to compromise a vehicle. They also identified weaknesses and exploited a PassThru device, used for servicing and diagnostics by dealerships. They have shown that a malicious PassThru device can be used to send CAN messages to a vehicle and install malware onto the car's telematics unit.

Miller et al. [17] provide an extensive analysis of the wireless interfaces of a Jeep Cherokee as potential attack surfaces. Most notably, they took advantage of vulnerabilities in the Jeep's UConnect system, which provides a cellular connection to the vehicle, and showed how they could completely control the vehicle over the Internet. They were able to control the car's dashboard functions, steering, brakes, heating system, radio, windshield wipers, the car's digital display and transmission. They demonstrated the attacks live, on the road, for Wired [8].

Ultimately, these attacks all rely on the fact that messages can be sent on the CAN network by a malicious attacker or a compromised ECU, and they are accepted by all other ECUs as if they were legitimate. The lack of source authentication is an enabler for all these types of attacks. While vehicles are designed to tolerate random failures, they cannot currently cope with malicious cyber-attacks. The lock-down of components is not a viable solution, both from a legislative point of view (e.g., right-to-repair legislation) or from the economic point of view of the manufacturer.

The AUTOSAR [1] specifications are a set of standards for ECU software functionality. The purpose of the standard is to reduce the development cost of ECU software and increase its scalability. The 4.2 release of the specification includes provisions for security on CAN. It provides interfaces and guidelines for authentication of messages, but leaves the implementation up to the manufacturer. The documents introduce the *Secure On-board Communication* module and provide guidelines for implementing authentication. They recommend using 128-bit keys, 64-bit MACs, and counters or timestamps to provide freshness. The MAC computation should be based on the data identifier, the data to be sent and the freshness value.

Our contribution. This paper proposes LeiA, the first AUTOSAR compliant, lightweight authentication protocol in the literature. The protocol respects the requirements laid out to become a standard in the automotive industry, as described in the Secure On-board Communication Module Specification, AUTOSAR Release 4.2.

LeiA does not require additional hardware components or substantial implementation costs thus is less expensive than previously proposed solutions, while providing higher security levels. The protocol has been designed by taking into consideration real-world requirements and limitations of the CAN bus such as limited bandwidth, short data frames and publisher-subscriber broadcast architecture where newly arrived messages overwrite older ones in the receiver's buffer.

Furthermore, LeiA is fully backwards compatible with existing CAN configuration, and is designed such that it can be flexibly implemented, providing different security vs bandwidth, computational overhead trade-offs.

Finally, we have proven the protocol to provide secure authentication under the unforgeability assumption of the MAC scheme under chosen plaintext attacks. Since we use the same MAC scheme for key diversification, we have the additional requirement that the produced MAC values are indistinguishable from the output of the key generation function.

Related work. CANAuth [19] and LiBrA-CAN [9] are two protocols for lightweight authentication over CAN. Both solutions make use of the CAN+ protocol, an improvement of the existing CAN. The CAN+ protocol was introduced by Ziermann et al. in [23]. It takes advantage of the fact that additional data can be sent in time intervals where the nodes conforming to the original CAN protocol do not listen. Therefore, CAN+ is backwards compatible and allows CAN-conform nodes to operate undisturbed alongside CAN+ nodes. This solution allows the transmission of 16 CAN+ bits, for one CAN bit transmitted.

Both solutions require replacing the CAN transceivers, and therefore imply a large cost for the manufacturers. Also, the logistics that would be involved in upgrading vehicles already in use are unclear.

Several solutions which do not require modified hardware have been proposed. We discuss them below and highlight their differences in respect with our proposed protocol, LeiA.

MaCAN is an authenticated protocol described in [10]. It is designed specifically for the CAN bus and takes into account the network's constraints, such as message length and available resources. MaCAN authenticates 4-byte messages with 4-byte MACs, in bidirectional communication. Timestamps are used as source of freshness, therefore, a *time server* is added into the system, which broadcasts a timestamp at regular intervals. Also, a *key server* is added, which shares a symmetric long term key with each security-enabled node. The key server mediates the establishment of keys between two nodes that want to securely communicate. In the case multiple nodes need to be able to verify the authenticity of a message, they propose using group keys. Bruni et al. give a formal analysis of the MaCAN protocol in [3]. They formally prove the secrecy of both long term keys and session keys used by the protocol. However, they found an attack through which one node is left unauthenticated and proposed a corrected version of the protocol. MaCAN introduces two new elements in the network, a time server and a key server. In LeiA, we remove the need for these components by using counters, instead of the time server, and by having each node derive the session keys locally, instead of having a key server.

LCAP was proposed by Hazem et al. in [11] and is a lightweight broadcast authentication protocol, which closely follows the CAN specification. The authors propose the use of a "magic number", which is appended to the message, instead of MACs. The number is part of a chain, which is obtained by applying a transformation function on an initial value, multiple times. Given the end of the chain, the sender and receiver can both verify if a value belongs to it. The magic number is 2 bytes in length. Handshakes are used in order to establish the secure channel and keep the nodes synchronised. This requires a significant number of CAN message identifiers be added to the network (five new IDs for each sender-receiver pair). The advantage of LCAP is that it only uses 2 bytes of the payload in order to achieve the authentication property, thus having a small overhead for authenticated message exchange. However, due to the high number of new IDs to be introduced in the network configuration, LCAP requires a large address space. Also, the channel setup and soft/hard synchronisation functions require a significant number of messages to be exchanged, thus adding to the overhead.

CaCAN has been introduced in [16], by Kurachi et al. Their approach is to use a *monitor node*, which authenticates the other nodes in the network. It detects and destroys unauthorised data frames by overwriting them with an error frame in real time. Challenge-response authentication is used in order to establish the secure channel. This approach requires a modified CAN controller, the monitor node, to be fitted in every car. Also, as is the general case with centralised authorities, if the monitor node is compromised or removed, the entire network is compromised as well.

Overview. This paper is organised as follows. Section 2 introduces standard security definitions, most of it is (adapted) from the literature. Section 3 provides the design and specification of our protocol. We give a formal security evaluation in Section 4. In Section 5 we discuss how we deal with the shortcomings of CAN and we provide guidelines for implementing the protocol in practise. We conclude in Section 6.

2 Security Notions and Adversarial Model

An authentication protocol is an interactive cryptographic protocol executed between a prover $\mathcal{P}$ and a verifier $\mathcal{V}$. In an initial phase, both parties run a $\texttt{setup}(\eta)$ function, which produces a shared secret s and, potentially, public parameters ns. After an execution of the protocol, $\mathcal{V}$ outputs the identity of the prover, id, and the message $data$. We say that the protocol has completeness error α if for all secrets s generated by $\texttt{setup}(\eta)$, the honestly executed protocol rejects the identity and message with a probability at most α.

We will show that our protocol is secure against active attacks. These allow the adversary $\mathcal{A}$ to interact with the honest prover a polynomial amount of times. Then, $\mathcal{A}$ interacts with the verifier only, and wins if the verifier returns $\texttt{accept}$. The adversary interacts with $\mathcal{V}$ only once. An authentication protocol is (t, Q, η)-*secure against active adversaries* if every probabilistic polynomial time (PPT) adversary $\mathcal{A}$, running at most t times and making Q queries to the honest prover, has probability at most ε to win the above game.

We first need to introduce some notation. Let $\mathbb{F}_2 = \{\mathtt{0}, \mathtt{1}\}$ be the field of two elements (or the set of Booleans). $\mathbb{F}_2^l$ denotes a bitstring of length l and $\mathbb{F}_2^*$ is a bitstring of arbitrary length. $\|$ stands for the concatenation of two bitstrings.

Execution environment. Let n be the number of identifiers in the system, and $\mathcal{I} = \{\mathsf{id}_0, \ldots, \mathsf{id}_{n-1}\}$ be the set of all identifiers. Let $\mathsf{P} = \{P_0, \ldots, P_{n-1}\}$ be the set of all protocol participants, where participant P_i knows the secret parameter s_i and public parameters ns.

Definition 1 (Protocol setup). Let the function $\texttt{setup} : \eta \to (s, ns)$ be the initialisation procedure of the protocol parties, where η is the security parameter and (s, ns) is a tuple formed by the secret parameter s and the public parameters ns.

Definition 2 (Authentication oracles). Let $\mathit{\Pi} = \{\pi(s_i) \mid s_i \in s\}$ be a set of oracles such that $\pi(s_i)$ emulates party P_i of the authentication protocol.

Definition 3 (Protocol output). Let $\texttt{output}\colon \mathsf{P} \to \mathcal{I} \times \mathbb{F}_2^*$ be the protocol output function of a protocol participant P_i and outputs a tuple $(\mathsf{id}_j, data)$ corresponding to the last successful protocol instance of P_i, where $\mathsf{id}_j \in \mathcal{I}$ is the identity of sender and $data$ is the message that was sent.

We will now introduce the security notions for symmetric key authentication protocols. Most of it is standard, most of the definitions proposed here are adapted from [20].

Definition 4 (Matching conversations [20]). We define *matching conversations* as a successful execution of the authentication protocol, between two parties.

We introduce the authentication game $\mathbf{Auth}_{\Pi}(\eta, \mathcal{A})$ and give a formal definition below. The public and secret parameters are generated by calling the $\mathtt{setup}(\eta)$ function. Then adversary $\mathcal{A}$ interacts with the oracles $\pi(s_i)$ which emulate the protocol participants which respond according to the protocol description. At some point the adversary $\mathcal{A}$ terminates. $\mathcal{A}$ wins if there is a party P_i which has accepted, and thus outputs, $(\mathsf{id}_j, data)$ while P_i and P_j did not have any matching conversation.

We denote by $\mathsf{Adv}^{\mathrm{Auth}}_{\mathsf{MAC}}(\eta, \mathcal{A})$ the advantage of the adversary $\mathcal{A}$ in breaking the authentication protocol.

```
Experiment $\mathbf{Auth}_{\Pi}(\eta, \mathcal{A})$
  $ns, s \leftarrow \mathtt{setup}(\eta)$
  $\mathcal{A}^{\Pi(ns,s)}(\eta, ns)$
  winif $\exists\ i, j, data : \mathtt{output}(P_i) = (\mathsf{id}_j, data)$ is the output of
  a party $P_i$ and, parties $P_i$ and $P_j$ did not have any matching
  conversation.
```

Definition 5 (Authentication Protocol Security). An authentication protocol is said to be *secure* if for all PPT adversaries $\mathcal{A}$, the probability that $\mathcal{A}$ wins the game $\mathbf{Auth}_{\Pi}(\eta, \mathcal{A})$ is a negligible function of η:

$$\mathsf{Adv}^{\mathrm{Auth}}_{\mathsf{MAC}}(\eta, \mathcal{A}) \leq \varepsilon(\eta)$$

Message Authentication Codes

A message authentication code is a set of three algorithms $\{\mathsf{KG}, \mathsf{MAC}, \mathsf{Verify}\}$, with associated key space $\mathcal{K}$, message space $\mathcal{M}$ and MAC space Φ.

The standard security notion for a MAC is *unforgeability under a chosen message attack (uf-cma)*. The secret key K is generated by calling the key generation algorithm KG of the MAC. Then, adversary $\mathcal{B}$ makes up to Q queries to the $\mathsf{MAC}(K, \cdot)$ and $\mathsf{Verify}(K, \cdot, \cdot)$ algorithms. At some point, $\mathcal{B}$ terminates and outputs a tuple $(\mathbf{m}, \phi)$, where $\mathbf{m} \in \mathcal{M}$ is a message and $\phi \in \Phi$ is a MAC. Adversary $\mathcal{B}$ wins if it did not query $\mathsf{MAC}(K, \mathbf{m})$ and ϕ verifies for message $\mathbf{m}$, under the secret key K.

We denote by $\mathsf{Adv}^{\mathrm{uf-cma}}_{\mathsf{MAC}}(\eta, \mathcal{B}, Q)$ the advantage of the adversary $\mathcal{B}$ in forging a messaged under a chosen message attack for MAC, on the security parameter η.

```
Experiment $\mathbf{UF\text{–}CMA}_{\mathsf{MAC}}(\eta, \mathcal{B}, Q)$
    $K \leftarrow \mathsf{KG}(1^\eta)$
    Invoke $\mathcal{B}^{\mathsf{MAC}(K,\cdot),\mathsf{Verify}(K,\cdot,\cdot)}$ which can make up to $Q$ queries
    to $\mathsf{MAC}(K,\cdot)$ and $\mathsf{Verify}(K,\cdot,\cdot)$.
    $(\mathbf{m}, \phi) \leftarrow \mathcal{B}^{\mathsf{MAC}(K,\cdot),\mathsf{Verify}(K,\cdot,\cdot)}$
    winif
        1. $\mathsf{Verify}(K, \mathbf{m}, \phi) = \mathtt{accept}$
        2. $\mathcal{B}$ did not already request $\mathsf{MAC}(K, \mathbf{m})$
```

Definition 6 (UF–CMA Security). We say that MAC is (t, Q, η)-secure against uf–cma adversaries if for any adversary $\mathcal{B}$ running in time t the experiment above, we have:

$$\mathsf{Adv}^{\text{uf–cma}}_{\mathsf{MAC}}(\eta, \mathcal{B}, Q) \leq \varepsilon(\eta)$$

Assumption 1 (MAC indistinguishability from random). We assume that the output of the MAC algorithm is computationally *indistinguishable from random* and, the output of the key generation (KG) function of the MAC algorithm and the output of the MAC function have the same distribution.

Adversarial model. We consider a Dolev-Yao adversary [6], who controls the network. In particular, she can passively monitor the network, reading all data passing through the CAN and send messages with any id. She can also send error frames to destroy current data or remote frames. However, in practise, the CAN error handling limits the attacker's capabilities in this respect.

3 LeiA: A Lightweight Authentication Protocol for CAN

This section outlines the design of LeiA, with a detailed description of each function of the authentication protocol.

The CAN bus uses a publish-and-subscribe architecture model, where one ECU can broadcast a message with a certain identifier (id_i). The identifier is not a way to identify the source or destination of a message, therefore, our protocol provides unidirectional authentication, with a method of signalling if any of the subscribed ECUs have gone out of sync/authentication failed.

Each protocol participant which needs to authenticate data, will need to store a tuple $\langle \mathsf{id}_i, K_{\mathsf{id}_i}, e_{\mathsf{id}_i}, K^e_{\mathsf{id}_i}, c_{\mathsf{id}_i} \rangle$ per relevant CAN identifier, where:

- the identifier id_i is a CAN ID;
- the key K_{id_i} is a 128-bit long term symmetric key that is used to derive the session key;
- the epoch e_{id_i} is a 56-bit counter; the value is incremented at every vehicle start-up or when the counter c_{id_i} overflows; participates in the generation of the session key;
- the session key $K^e_{\mathsf{id}_i}$ is a 128-bit key used for generating the MAC; regenerating the session key when the epoch e_{id_i} changes ensures that only

a small amount of data is authenticated under the same key; also, if the session key becomes compromised, the attacker can compute valid MACs only until the epoch changes (limited time);
– the counter c_{id_i} is a 16-bit counter included in the Message Authentication Code (MAC) and is sent within the Data Frame containing the MAC, in order to provide freshness.

The long term keys and epochs are assumed to be stored in tamper-resistant memory. Updating the set of keys (e.g. if adding or replacing a node in the network) should require direct physical access to the involved nodes and, therefore, could only be done by an authorised repairs shop. How exactly this is done is beyond the scope of this paper.

We describe below the functions of the protocol for a pair of nodes: sender S, which is the broadcaster of messages with the identifier id_i, and receiver R, which is the node subscribed to messages broadcast on the identifier id_i.

The authentication protocol LeiA has an associated key space $\mathcal{K} \in \mathbb{F}_2^{128}$, message space $\mathcal{M} \in \mathbb{F}_2^{*}$ and MAC space $\Phi \in \mathbb{F}_2^{64}$.

Protocol setup. The function `setup`: $\eta \rightarrow (s, ns)$ is the initialisation procedure of the ECUs, where η is the security parameters and (s, ns) is a tuple formed by the secret parameter s and the public parameters ns. The secret parameter $s = \langle K_{\mathsf{id}_0}, \ldots K_{\mathsf{id}_{n-1}} \rangle$ is computed by running the key generation algorithm $\mathsf{KG}(1^{\eta})$ of the MAC for each identity id_i, with $K_{\mathsf{id}_i} \in \mathcal{K}$. The public parameters are $ns = \big\langle (c_{\mathsf{id}_0}, e_{\mathsf{id}_0}), \ldots, (c_{\mathsf{id}_{n-1}}, e_{\mathsf{id}_{n-1}}) \big\rangle$, where $c_{\mathsf{id}_i} \in \mathbb{F}_2^{16}$ is the counter and $e_{\mathsf{id}_i} \in \mathbb{F}_2^{56}$ is the epoch. Both the counter and epoch are initialised to zero, for each identity id_i. The session key generation function is then called for each identity id_i, in order to generate the session key $K_{\mathsf{id}_i}^{e}$.

Session key generation (Figure 1)**.**
Let `session_key_gen`: $\mathcal{K} \times \mathbb{F}_2^{56} \rightarrow \mathcal{K}$ be the session key generation function. This function takes as input a long term symmetric key K_{id_i} and an epoch e_{id_i}, both associated with an identity id_i, and outputs the session key $K_{\mathsf{id}_i}^{e}$ computed as follows:

1. increment epoch: $e_{\mathsf{id}_i} \leftarrow e_{\mathsf{id}_i} + 1$
2. apply the MAC algorithm on the epoch:

$$K_{\mathsf{id}_i}^{e} \leftarrow \mathsf{MAC}(K_{\mathsf{id}_i}, e_{\mathsf{id}_i})$$

3. reset counter to zero: $c_{\mathsf{id}_i} \leftarrow 0$

Sending authenticated messages (Figure 2)**.**

In order to send an authenticated message, the sender first needs to update the counter c_{id_i}. If c_{id_i} overflows, then the epoch e_{id_i} is incremented and c_{id_i} is reset to 0 (see Algorithm 1). It then calls the MAC algorithm which takes as

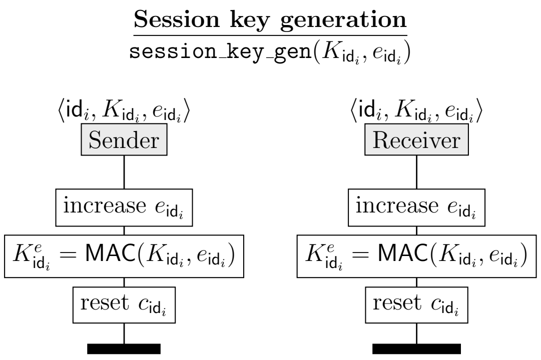

Session key generation
session_key_gen($K_{\mathsf{id}_i}, e_{\mathsf{id}_i}$)
$\langle \mathsf{id}_i, K_{\mathsf{id}_i}, e_{\mathsf{id}_i} \rangle$
Sender
$\langle \mathsf{id}_i, K_{\mathsf{id}_i}, e_{\mathsf{id}_i} \rangle$
Receiver
increase e_{id_i}
$K^e_{\mathsf{id}_i} = \mathsf{MAC}(K_{\mathsf{id}_i}, e_{\mathsf{id}_i})$
reset c_{id_i}

Figure 1. Session key generation between sender S and receiver R for message with identifier id_i.

input the session key $K^e_{\mathsf{id}_i}$, the counter c_{id_i} and the message *data*, and produces as output a MAC $\phi \in \Phi$ computed as:

$$\phi = \mathsf{MAC}(K^e_{\mathsf{id}_i}, c_{\mathsf{id}_i}, data)$$

The sender then transmits the counter, data and MAC. After reading the values, the receiver updates the counters and verifies the MAC.

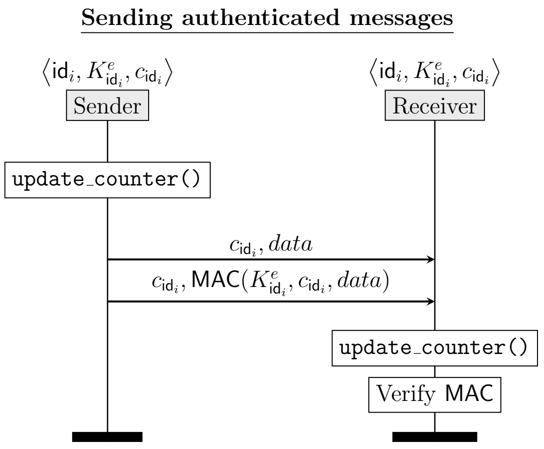
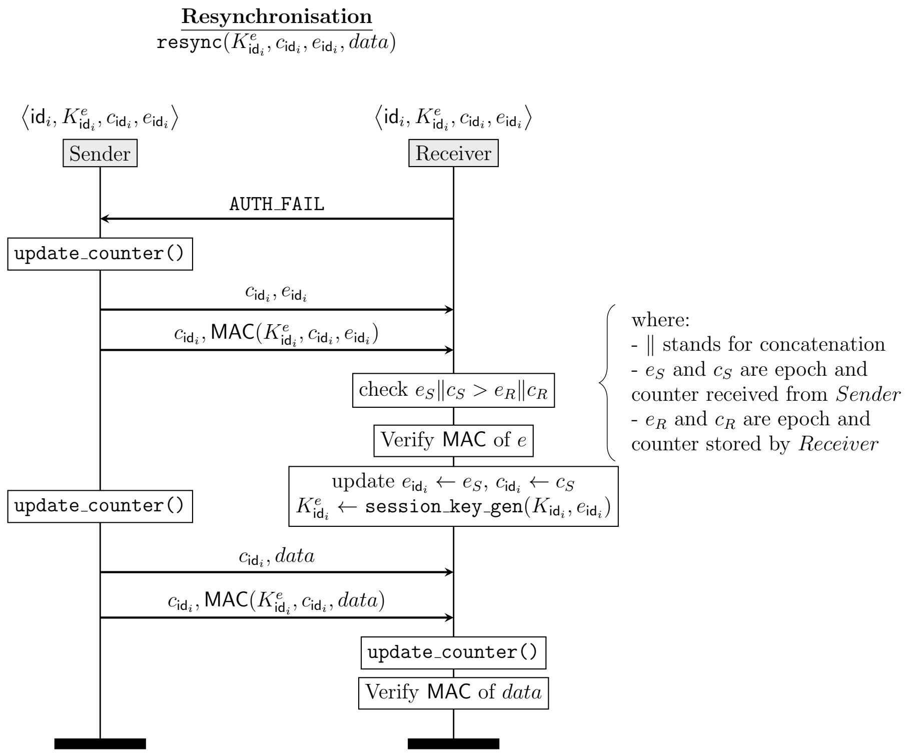

Sending authenticated messages
$\langle \mathsf{id}_i, K^e_{\mathsf{id}_i}, c_{\mathsf{id}_i} \rangle$
Sender
$\langle \mathsf{id}_i, K^e_{\mathsf{id}_i}, c_{\mathsf{id}_i} \rangle$
Receiver
update_counter()
$c_{\mathsf{id}_i}, data$
$c_{\mathsf{id}_i}, \mathsf{MAC}(K^e_{\mathsf{id}_i}, c_{\mathsf{id}_i}, data)$
update_counter()
Verify MAC

Figure 2. Message authentication between sender S and receiver R for message with identifier id_i.

Resynchronisation (Figure 3)**.**
If a MAC cannot be verified, the receiver sends an `AUTH_FAIL` signal to the sender. When an `AUTH_FAIL` message is read, the sender S broadcasts a message containing its current epoch value, a MAC of the epoch and counter c_{id_i},

Algorithm 1 `update_counter()` function

```
Require: counter $c_{\mathsf{id}_i}$, epoch $e_{\mathsf{id}_i}$, LTSK $K_{\mathsf{id}_i}$
Ensure: $c_{\mathsf{id}_i}$ and $e_{\mathsf{id}_i}$ are incremented accordingly
 1: if $c_{\mathsf{id}_i} = \texttt{0xFFFF}$ then
 2:     if $e_{\mathsf{id}_i} = \texttt{0xFFFFFFFFFFFFFF}$ then
 3:         $e_{\mathsf{id}_i} \leftarrow \texttt{0x00000000000000}$
 4:     else
 5:         $e_{\mathsf{id}_i} \leftarrow e_{\mathsf{id}_i} + 1$
 6:     end if
 7:     $c_{\mathsf{id}_i} \leftarrow \texttt{0x0000}$
 8:     call session_key_gen($K_{\mathsf{id}_i}, e_{\mathsf{id}_i}$)
 9: else
10:     $c_{\mathsf{id}_i} \leftarrow c_{\mathsf{id}_i} + 1$
11: end if
```



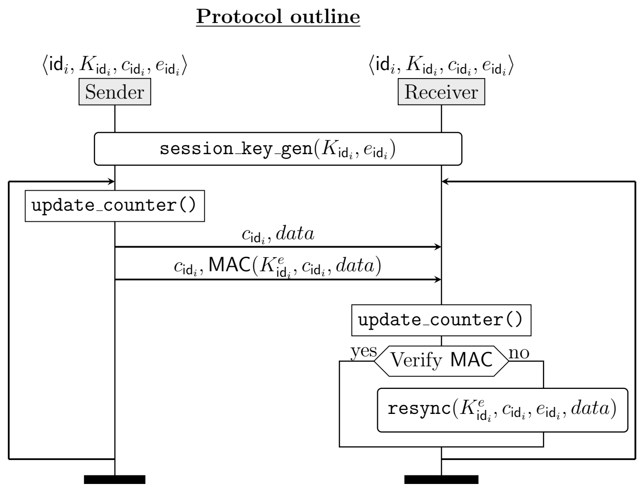
Resynchronisation
$\texttt{resync}(K^e_{\mathsf{id}_i}, c_{\mathsf{id}_i}, e_{\mathsf{id}_i}, data)$
$\langle \mathsf{id}_i, K^e_{\mathsf{id}_i}, c_{\mathsf{id}_i}, e_{\mathsf{id}_i} \rangle$
Sender
$\langle \mathsf{id}_i, K^e_{\mathsf{id}_i}, c_{\mathsf{id}_i}, e_{\mathsf{id}_i} \rangle$
Receiver
AUTH_FAIL
update_counter()
$c_{\mathsf{id}_i}, e_{\mathsf{id}_i}$
$c_{\mathsf{id}_i}, \mathsf{MAC}(K^e_{\mathsf{id}_i}, c_{\mathsf{id}_i}, e_{\mathsf{id}_i})$
check $e_S \| c_S > e_R \| c_R$
Verify MAC of e
update $e_{\mathsf{id}_i} \leftarrow e_S$, $c_{\mathsf{id}_i} \leftarrow c_S$
$K^e_{\mathsf{id}_i} \leftarrow \texttt{session\_key\_gen}(K_{\mathsf{id}_i}, e_{\mathsf{id}_i})$
update_counter()
$c_{\mathsf{id}_i}, data$
$c_{\mathsf{id}_i}, \mathsf{MAC}(K^e_{\mathsf{id}_i}, c_{\mathsf{id}_i}, data)$
update_counter()
Verify MAC of $data$
where:
- $\|$ stands for concatenation
- e_S and c_S are epoch and counter received from *Sender*
- e_R and c_R are epoch and counter stored by *Receiver*

Figure 3. Message authentication failure and resynchronisation procedure, between sender S and receiver R for message with identifier id_i.

then proceeds with normal data transmission. This will help the receiver nodes resynchronise their epoch and counter.

R will only update e_{id_i} and c_{id_i} if the values are higher (e_{id_i} received can be equal to e_{id_i} stored) than the stored ones. If the new counter is lower than the receiver's counter, it means there is an attacker performing a replay attack, therefore the data is discarded and the counter not incremented.

Most common cause for a MAC to fail verification, in the context of the CAN, is the de-synchronisation of counter c_{id_i} and epoch e_{id_i} values. Not all nodes join the network at the same time, therefore the counters will be outdated and the receiver will need to request the current values from the sender. A complete protocol outline is given in Figure 4.



Protocol outline
$\langle \mathsf{id}_i, K_{\mathsf{id}_i}, c_{\mathsf{id}_i}, e_{\mathsf{id}_i} \rangle$
Sender
$\langle \mathsf{id}_i, K_{\mathsf{id}_i}, c_{\mathsf{id}_i}, e_{\mathsf{id}_i} \rangle$
Receiver
session_key_gen($K_{\mathsf{id}_i}, e_{\mathsf{id}_i}$)
update_counter()
$c_{\mathsf{id}_i}, data$
$c_{\mathsf{id}_i}, \mathsf{MAC}(K^e_{\mathsf{id}_i}, c_{\mathsf{id}_i}, data)$
update_counter()
yes
Verify MAC
no
resync($K^e_{\mathsf{id}_i}, c_{\mathsf{id}_i}, e_{\mathsf{id}_i}, data$)

Figure 4. Communication between sender S and receiver R for message with identifier id_i – LeiA protocol outline: first, the session keys are generated by both participants; then, S can send authenticated message to R; R verifies the MAC of the received message; if the verification fails, the resynchronisation is initialised, otherwise, the message is accepted.

4 Security Analysis

This section analyses the security of LeiA under the unforgeability assumption of the MAC scheme under chosen message attacks.

Theorem 2. *The* LeiA *authentication protocol is secure with respect to Definition 5 (see Section 2).*

Proof. Assume that there is an adversary $\mathcal{A}$ that breaks the $\mathbf{Auth}_{\Pi}(\eta, \mathcal{A})$ security of the authentication protocol LeiA. Then, we build an adversary $\mathcal{B}$ that breaks the (t, Q, η)-security of the $\mathbf{UF\text{–}CMA}_{\mathsf{MAC}}$ scheme.

At the beginning, the adversary $\mathcal{B}$ randomly picks one target identifier $\mathsf{id}^\star$ and a target epoch $e^\star$. Then, $\mathcal{B}$ runs the protocol setup function for each identity id_i.

The adversary $\mathcal{B}$ executes $\mathcal{A}$. For this, $\mathcal{B}$ needs to emulate oracles $\pi(K_{\mathsf{id}_i})$. Emulating party P_i means generating the session key, and keeping track of the counters c_{id_i} and epochs e_{id_i}, as specified in the protocol description. The session key for an identity is regenerated every time the associated epoch is incremented. The adversary $\mathcal{A}$ has access to the oracles in Π.

When transitioning from $e^\star - 1$ to $e^\star$, for identity $\mathsf{id}^\star$, $\mathcal{B}$ will not use the MAC algorithm, as described in the protocol, to generate the session key $K^{e^\star}_{\mathsf{id}^\star}$. Instead, whenever a MAC needs to be computed under the key $K^{e^\star}_{\mathsf{id}^\star}$, the adversary will use the $\mathsf{MAC}(\cdot,\cdot)$ oracle from the $\mathbf{UF\text{–}CMA}_{\mathsf{MAC}}$ game. Note that due to Assumption 1, this will be indistinguishable from the case of using the key generation algorithm $\mathsf{KG}(\cdot)$. For all other cases, it will compute it herself, by running the MAC algorithm.

At some point, $\mathcal{A}$ terminates. With non-negligible probability, there must exist a P_i which outputs an identity id_j and a message $\mathbf{m}$, without having a matching conversation between P_i and P_j. In order for P_i to produce this output, it means $\mathcal{A}$ has sent a message $\mathbf{m} = (c\|data)$ and a MAC $\phi = \mathsf{MAC}(K^{e}_{\mathsf{id}}, \mathbf{m})$ which P_i has verified, and therefore this must be a valid MAC.

If $\mathsf{id}_j = \mathsf{id}^\star$ and $e = e^\star$, the adversary $\mathcal{B}$ will output $(\mathbf{m}, \phi)$; otherwise, it will output a tuple of random strings. As the identity $\mathsf{id}^\star$ and epoch $e^\star$ are chosen at random before the $\mathtt{setup}(\eta)$ phase, the probability that $\mathcal{A}$ also attacks $\mathsf{id}^\star$ and $e^\star$ is:

$$\mathcal{P}(K^{e^\star}_{\mathsf{id}^\star} = K^{e}_{\mathsf{id}_j}) = \frac{1}{n} \cdot \frac{1}{2^{56}}$$

and we recall that n is the number of identifiers in the system.

In order to win the $\mathbf{UF\text{–}CMA}_{\mathsf{MAC}}$ game, the adversary needs:

1. $\mathsf{Verify}(K^{e^\star}_{\mathsf{id}^\star}, \mathbf{m}, \phi) = \mathtt{accept}$;
2. the MAC ϕ was never queried to the MAC oracle.

Condition 1. holds because ϕ is a valid MAC, as it was verified by party P_i. Condition 2. holds because the MAC was never queried to the MAC oracle, as P_i and P_j do not have a matching conversation. □

5 Dealing with the Shortcomings of CAN

As some of the ECUs are involved in safety-critical functions such as acceleration and ABS, latency is of prime concern. Any solution aiming at providing extra security features, such as authentication, cannot introduce significant latency. To this end, lightweight cryptography is best suited. Furthermore, many ECUs have limited memory available, therefore the implementation of the protocol should be compact as well. For this reason, our solution uses a MAC algorithm for two different purposes: authenticating data and deriving session keys.

In order to compensate for the modest security provided by lightweight cryptographic primitives, we do not use the long term secret key directly, but generate session keys, which are used to authenticate the messages exchanged. A session key is used to authenticate at most 2^{16} messages, after which a new session key is derived. This limits the amount of key-dependent data an attacker has access to. In case a session key is compromised, an attacker can use it either until 2^{16} messages have been authenticated, or until the vehicle is restarted, whichever comes first.

LeiA makes use of the extended identifier data frames. It uses the Extended Identifier 18-bit field in order to send the 16-bit counter and a 2-bit command code, as explained below (Figure 5). The 29-bit identifier data frames co-exist with the 11-bit data frames without interfering with the arbitration process of CAN, as the priority of a message is decided based on the 11-bit Identifier field.

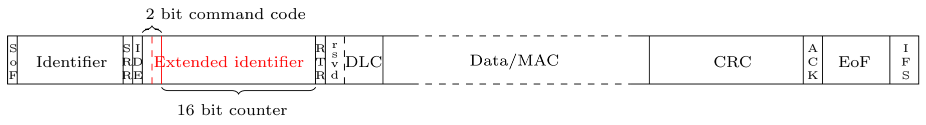


Figure 5. Extended Data Frame CAN 2.0B (29-bit identifier) – placement of command code and counter within Extended Identifier field.

We define three transmission channels over CAN:

Data Channel

All ids which are used to transmit data and signals constitute the data channel. The data is transmitted within the payload field of the frame. The counter c_{id_i} which is used to generate the MAC is placed in the extended identifier field. The two leftmost bits are the *command code* `00`, and signal that data is being transmitted in the frame.

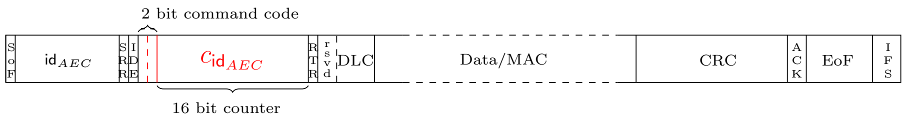
Authentication Channel

All ids which are used to transmit MACs make up the Authentication Channel. The MACs are transmitted on a different identifier than the data. We propose this id be a fixed offset from the base id on which the data is sent. It should be as close as possible to the base id, in order to avoid scheduling issues caused by arbitration. In our example, $\mathsf{id}_{MAC} = \mathsf{id}_{data} + 1$. This will avoid messages with the same identifier being overwritten in the CAN controller *buffer*. The counter is placed in the extended identifier field. The two leftmost bits, which represent the *command code*, are defined as follows:

01: the data frame contains a MAC of data;
10: the data frame contains an epoch value e_{id_i};
11: the data frame contains a MAC of an epoch e_{id_i}.

Authentication Error Channel (AEC)

Each node connected to CAN has an Authentication Error Channel, AEC. This is used for resynchronisation purposes. The `AUTH_FAIL` signal is sent on the AEC. Nodes which are broadcasters of messages with id_i become subscribers of the AEC of the nodes listening to id_i. The `AUTH_FAIL` signal is defined as a set of two messages. The first data frame contains the id of the message which failed MAC verification (id_{failed}), concatenated with the lower 53 bits of the AEC epoch counter ($lsb_{53}(e_{\mathsf{id}_{AEC}})$). Sending the epoch within the data frame ensures the receiving nodes can verify they have the correct values, and a resynchronisation procedure for the AEC is not needed. The second message contains the MAC of the previous one, as shown in Figure 6. Sending an `AUTH_FAIL` signal is considered a rare event, therefore overwriting messages within the buffer are not of concern, in contrast to data transmission. Thus, we can use the same identifier (id_{AEC}) for both message types.



where:

data = $\mathsf{id}_{failed} \| lsb_{53}(e_{\mathsf{id}_{AEC}})$

MAC = $MAC(K^{e}_{\mathsf{id}_{AEC}}, c_{\mathsf{id}_{AEC}}, \mathsf{id}_{failed}, lsb_{53}(e_{\mathsf{id}_{AEC}}))$

Figure 6. Data frame structure for `AUTH_FAIL` signal.

Table 1 shows a small example of an extended communication matrix. The *identifiers* highlighted are the additional identifiers introduced by LeiA. Identifiers `0x005`, `0x011` and `0x016` correspond to the *Authentication Channel*, while identifiers `0x7FD`, `0x7FE` and `0x7FF` correspond to the *Authentication Error Channel*.

Table 1. Extended communication matrix example. 'S' stands for Sender and 'R' for Receiver.

Identifier	Node A	Node B	Node C	Node D
id = 0x004	S		R	
id = 0x010		R	R	S
id = 0x015		S		R

→

Identifier	Node A	Node B	Node C	Node D
id = 0x004	S		R	
id = 0x005	S		R	
id = 0x010		R	R	S
id = 0x011		R	R	S
id = 0x015		S		R
id = 0x016		S		R
id = 0x7FD		S		R
id = 0x7FE	R		S	R
id = 0x7FF		R		S

The procedures of sending authenticated messages and re-synchronisation, complete with *command code* placement are shown in Figure 7 and Figure 8.

The CAN bus has a static configuration. Due to this, LeiA can be implemented in two ways, depending on the functionality of the ECU. As described above, the protocol requires each message to be accompanied by a MAC. If

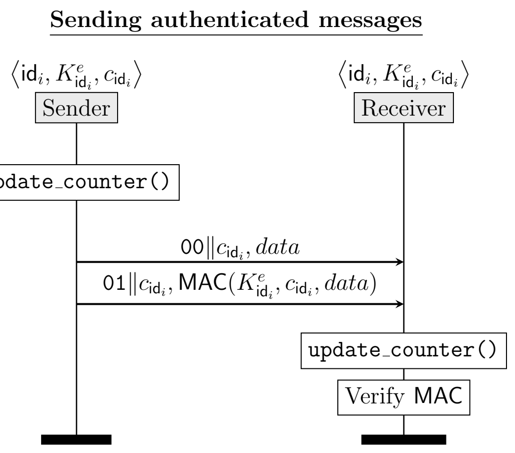

Sending authenticated messages
$\langle \mathsf{id}_i, K^e_{\mathsf{id}_i}, c_{\mathsf{id}_i} \rangle$
Sender
$\langle \mathsf{id}_i, K^e_{\mathsf{id}_i}, c_{\mathsf{id}_i} \rangle$
Receiver
update_counter()
$\mathsf{00} \| c_{\mathsf{id}_i}, data$
$\mathsf{01} \| c_{\mathsf{id}_i}, \mathsf{MAC}(K^e_{\mathsf{id}_i}, c_{\mathsf{id}_i}, data)$
update_counter()
Verify MAC

Figure 7. Message authentication between sender S and receiver R for message with identifier id_i, with command code.

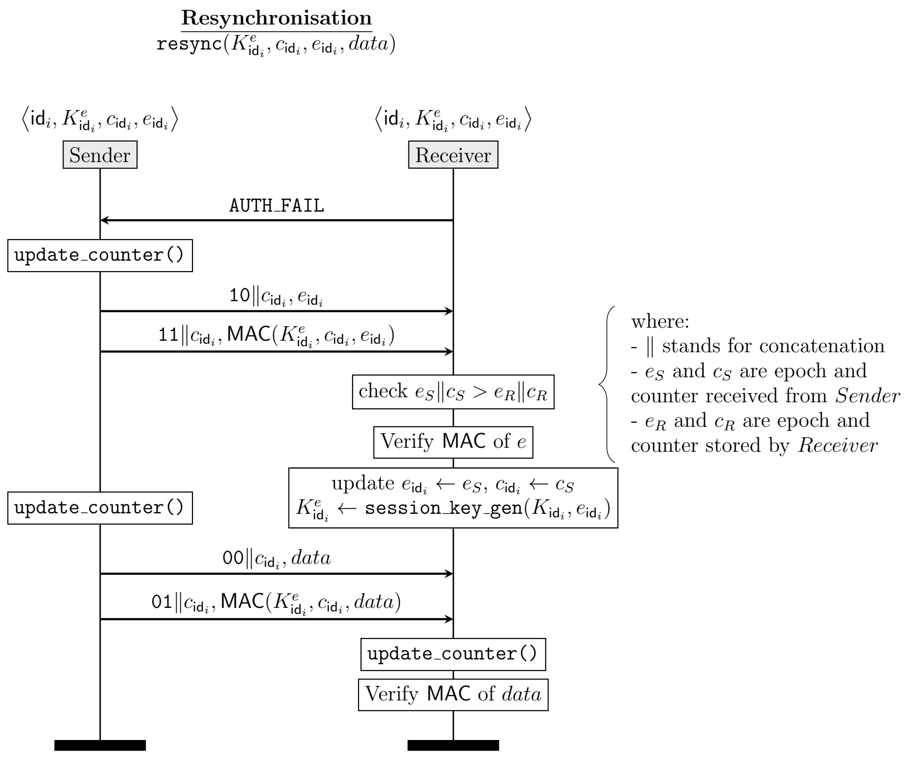

Resynchronisation
$\mathtt{resync}(K^e_{\mathsf{id}_i}, c_{\mathsf{id}_i}, e_{\mathsf{id}_i}, data)$
$\langle \mathsf{id}_i, K^e_{\mathsf{id}_i}, c_{\mathsf{id}_i}, e_{\mathsf{id}_i} \rangle$
Sender
$\langle \mathsf{id}_i, K^e_{\mathsf{id}_i}, c_{\mathsf{id}_i}, e_{\mathsf{id}_i} \rangle$
Receiver
AUTH_FAIL
update_counter()
$\mathsf{10} \| c_{\mathsf{id}_i}, e_{\mathsf{id}_i}$
$\mathsf{11} \| c_{\mathsf{id}_i}, \mathsf{MAC}(K^e_{\mathsf{id}_i}, c_{\mathsf{id}_i}, e_{\mathsf{id}_i})$
check $e_S \| c_S > e_R \| c_R$
Verify MAC of e
update $e_{\mathsf{id}_i} \leftarrow e_S$, $c_{\mathsf{id}_i} \leftarrow c_S$
$K^e_{\mathsf{id}_i} \leftarrow \mathtt{session\_key\_gen}(K_{\mathsf{id}_i}, e_{\mathsf{id}_i})$
update_counter()
$\mathsf{00} \| c_{\mathsf{id}_i}, data$
$\mathsf{01} \| c_{\mathsf{id}_i}, \mathsf{MAC}(K^e_{\mathsf{id}_i}, c_{\mathsf{id}_i}, data)$
update_counter()
Verify MAC of $data$
where:
- $\|$ stands for concatenation
- e_S and c_S are epoch and counter received from *Sender*
- e_R and c_R are epoch and counter stored by *Receiver*

Figure 8. Message authentication failure and resynchronisation procedure, between sender S and receiver R for message with identifier id_i, with command code placement.

applied to all ECUs, this doubles the communication overhead. However, for nodes not involved in safety-critical functions, the protocol can be implemented such that one MAC is sent after n messages, where n can be decided based on the node's security requirements. This allows manufacturers to choose a most suitable trade-off between security and bandwidth for their vehicles.

The CAN is an architecture which is highly susceptible to denial of service (DoS) attacks. LeiA is not a solution that tackles this issue, as it is out of the scope of our goals. However, DoS attacks do not affect the security of the protocol. In fact, under LeiA, messages that are not correctly authenticated are not parsed, saving ECUs time and computation energy.

In the case an attacker fully compromises and takes control of an ECU, for the ids the node broadcasts or listens on, the attacker will unavoidably be able to generate valid MACs, but not for any other id. This is not a problem of our protocol but an inherit limitation of using symmetric key cryptography.

An attacker can collect some `AUTH_FAIL` answers from the sender, knowing one of the receiver nodes is offline. When the receiver node joins the network and sends the `AUTH_FAIL` signal, as it does not have the correct counter and epoch values, the attacker sends a stored answer. The receiver will accept the message, provided the stored counter and epoch are lower than the received ones. However, due to the design of CAN, the initial `AUTH_FAIL` signal is also received by the sender node, which will send the correct epoch and counter values. The attacker can destroy these frames, but S will broadcast them again, due to the error handling mechanism of CAN. After a number of destroyed frames, the CAN flags the attacker as error passive, meaning it cannot destroy other frames. Therefore, the correct message of S will be transmitted and the receiver node will be able to update its values accordingly. Communication then resumes under the protocol.

We would like to emphasize that all other proposed authentication protocols from the literature are susceptible to DoS attacks and do not deal with attackers taking full control over an ECU.

Next we elaborate on how LeiA satisfies the requirements laid out by AUTOSAR 4.2, Secure On Board Communication Module. Regarding freshness, the specification states both sending and receiving sides need to maintain a Freshness Value (e.g. counter, timestamp). In LeiA, this is achieved by the 16-bit counters c_{id_i}, placed in the Extended Identifier field of a Data Frame. AUTOSAR recommends the use of 128-bit keys, which LeiA respects though K_{id}. It also states that, depending on the authentication algorithm chosen, the Message Authentication Code can be truncated, with a minimum recommended length of 64-bit. As described in our protocol, we use 64-bit MACs, which fit in the 8-byte Payload Field of a Data Frame. Furthermore, the standard requires the MAC to be calculated based on the id, data and complete freshness value. In LeiA, the MAC is computed based on the session key $K^e_{\mathsf{id}_i}$, which is uniquely associated with an identifier id_i, the counter c_{id_i} and the data to be transmitted. Regarding MAC verification failure, SecOC requires the receiver to attempt to verify for a number of times (defined by the parameter `SecOCFreshnessCounterSyncAttempts`), after which the data is dropped. LeiA uses the `resync` procedure, in order to keep the protocol in synch, and avoid a possible internal denial of service attack due to the de-synchronisation of counters.

6 Conclusion

We have proposed a new lightweight authentication protocol for CAN, LeiA, that allows ECUs to authenticate each other, therefore preventing a number of attacks presented in the literature. We have proven the protocol secure under the unforgeability assumption of the MAC scheme under a chosen message attack. LeiA has been designed to run under the stringent time and bandwidth constraints of automotive applications, and is backwards compatible with existing CAN configuration. LeiA is the first AUTOSAR compliant lightweight authentication protocol available in the literature. Also, our protocol achieves higher security levels than previously proposed solutions, without the need of additional hardware components or substantial implementation costs. Finally, we have taken into consideration the real-world requirements and constraints of the CAN bus, and discussed how we mitigated and overcame them. The properties of LeiA make it suitable for deployment in automotive applications as it strikes the right balance between practicality, cost, latency and security.

References

1. AUTOSAR: AUTOSAR Specification 4.2, http://www.autosar.org/specifications/release-42/
2. Bogdanov, A.: Linear Slide Attacks on the Keeloq Block Cipher. In: Information Security and Cryptology, Third SKLOIS Conference, Inscrypt 2007, Xining, China, August 31 - September 5, 2007, Revised Selected Papers. pp. 66–80 (2007)
3. Bruni, A., Sojka, M., Nielson, F., Nielson, H.R.: Formal Security Analysis of the MaCAN Protocol. In: Integrated Formal Methods. pp. 241–255. Springer (2014)
4. Checkoway, S., McCoy, D., Kantor, B., Anderson, D., Shacham, H., Savage, S., Koscher, K., Czeskis, A., Roesner, F., Kohno, T., others: Comprehensive Experimental Analyses of Automotive Attack Surfaces. In: 20th USENIX Security Symposium (USENIX Security 2011). San Francisco (2011)
5. Courtois, N., Bard, G.V., Wagner, D.: Algebraic and Slide Attacks on Keeloq. In: Fast Software Encryption, 15th International Workshop (FSE 2008), Lausanne, Switzerland, February 10-13, 2008, Revised Selected Papers. pp. 97–115 (2008)
6. Dolev, D., Yao, A.C.: On the Security of Public Key Protocols. Information Theory, IEEE Transactions on 29(2), 198–208 (1983)
7. Garcia, F.D., Oswald, D., Kasper, T., Pavlidès, P.: Lock it and Still Lose it - on the (In)security of Automotive Remote Keyless Entry Systems. In: 25nd USENIX Security Symposium (USENIX Security 2016), to appear. USENIX Association (2016)
8. Greenberg, A.: Hackers Remotely Kill a Jeep on the Highway – with me in it (2015), http://www.wired.com/2015/07/hackers-remotely-kill-jeep-highway/
9. Groza, B., Murvay, S., Van Herrewege, A., Verbauwhede, I.: LiBrA-CAN: A Lightweight Broadcast Authentication Protocol for Controller Area Networks. In: Cryptology and Network Security. pp. 185–200. Springer (2012)

10. Hartkopp, O., Reuber, C., Schilling, R.: MaCAN - Message Authenticated CAN. In: 10th Int. Conf. on Embedded Security in Cars (ESCAR 2012), Berlin, Germany. vol. 6 (2012)
11. Hazem, A., Fahmy, H.A.: LCAP - A Lightweight CAN Authentication Protocol for securing in-vehicle networks. In: 10th Int. Conf. on Embedded Security in Cars (ESCAR 2012), Berlin, Germany. vol. 6 (2012)
12. Indesteege, S., Keller, N., Dunkelman, O., Biham, E., Preneel, B.: A Practical Attack on Keeloq. In: Advances in Cryptology - EUROCRYPT 2008, 27th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Istanbul, Turkey, April 13-17, 2008. Proceedings. pp. 1–18 (2008)
13. ISO: 11898-1: 2003 - Road Vehicles - Controller Area Network. International Organization for Standardization, Geneva, Switzerland (2003)
14. Kasper, M., Kasper, T., Moradi, A., Paar, C.: Breaking Keeloq in a Flash: On Extracting Keys at Lightning Speed. In: Progress in Cryptology – AFRICACRYPT 2009: Second International Conference on Cryptology in Africa, Gammarth, Tunisia, June 21-25, 2009. Proceedings. pp. 403–420. Springer Berlin Heidelberg, Berlin, Heidelberg (2009)
15. Koscher, K., Czeskis, A., Roesner, F., Patel, S., Kohno, T., Checkoway, S., McCoy, D., Kantor, B., Anderson, D., Shacham, H., others: Experimental Security Analysis of a Modern Automobile. In: 31st IEEE Symposium on Security & Privacy (S & P 2010), on. pp. 447–462. IEEE (2010)
16. Kurachi, R., Matsubara, Y., Takada, H., Adachi, N., Miyashita, Y., Horihata, S.: CaCAN – Centralised Authentication System in CAN. In: 12th Int. Conf. on Embedded Security in Cars (ESCAR 2014) (2014)
17. Miller, C., Valasek, C.: Remote Exploitation of an Unaltered Passenger Vehicle (2015), http://illmatics.com/Remote%20Car%20Hacking.pdf
18. Studnia, I., Nicomette, V., Alata, E., Deswarte, Y., Kaâniche, M., Laarouchi, Y.: Survey on Security Threats and Protection Mechanisms in Embedded Automotive Networks. In: Dependable Systems and Networks Workshop (DSN-W 2013), 2013 43rd Annual IEEE/IFIP Conference on. pp. 1–12. IEEE (2013)
19. Van Herrewege, A., Singelee, D., Verbauwhede, I.: CANAuth - A Simple, Backward Compatible Broadcast Authentication Protocol for CAN bus. In: ECRYPT Workshop on Lightweight Cryptography 2011 (2011)
20. Vaudenay, S.: On Privacy Models for RFID. In: Advances in Cryptology–ASIACRYPT 2007, pp. 68–87. Springer (2007)
21. Verdult, R., Garcia, F.D.: Cryptanalysis of the Megamos Crypto Automotive Immobilizer. In: USENIX ;login:. vol. 40/6, pp. 17–22. USENIX Association (2015)
22. Verdult, R., Garcia, F.D., Balasch, J.: Gone in 360 Seconds: Hijacking with Hitag2. In: 21st USENIX Security Symposium (USENIX Security 2012). pp. 237–252 (2012)
23. Ziermann, T., Wildermann, S., Teich, J.: CAN+: A New Backward-compatible Controller Area Network (CAN) Protocol with up to 16x Higher Data Rates. In: Design, Automation & Test in Europe Conference & Exhibition (DATE 2009). pp. 1088–1093. IEEE (2009)
24. Verdult, R., Garcia, F.D., Ege, B.: Dismantling Megamos Crypto: Wirelessly Lockpicking a Vehicle Immobilizer. In: 22nd USENIX Security Symposium (USENIX Security 2013). pp. 703–718. USENIX Association (2015)