LOKI: A Lightweight Cryptographic Key Distribution Protocol for Controller Area Networks

Teri Lenard, Roland Bolboacă and Béla Genge
George Emil Palade University of Medicine, Pharmacy, Science and Technology of Târgu Mureş, Romania
Gheorghe Marinescu, No. 38, Târgu Mureş, Mureş, Romania, 540139
Email: teri.lenard@umfst.ro, roland.bolboaca@umfst.ro, bela.genge@umfst.ro

***Abstract*—The recent advancement in the automotive sector has led to a technological explosion. As a result, the modern car provides a wide range of features supported by state of the art hardware and software. Unfortunately, while this is the case of most major components, in the same vehicle we find dozens of sensors and sub-systems built over legacy hardware and software with limited computational capabilities. This paper presents *LOKI*, a lightweight cryptographic key distribution scheme applicable in the case of the classical in-vehicle communication systems. The *LOKI* protocol stands out compared to already proposed protocols in the literature due to its ability to use only a single broadcast message to initiate the generation of a new cryptographic key across a group of nodes. It's lightweight key derivation algorithm takes advantage of a reverse hash chain traversal algorithm to generate fresh session keys. Experimental results consisting of a laboratory-scale system based on Vector Informatik's CANoe simulation environment demonstrate the effectiveness of the developed methodology and its seamless impact manifested on the network.**

***Index Terms*—Security protocols, key management, controller area networks.**

I. Introduction

In the automotive sector the recent technological advancement and the need for more advanced features has led to a technological explosion. The main result is that the modern car provides a wide range of features supported by state of the art hardware and software. Unfortunately, while this is the case for most major components, in the same vehicle we find dozens of sensors and sub-systems built over legacy hardware and software with limited computational capabilities. In this context, the integration of state of the art security mechanisms becomes a challenging task, which requires innovative approaches.

In today's modern vehicles the "backbone" communication is provided by the Controller Area Network (CAN). Standardised in 2003 [1], it is an International Standardization Organization (ISO) - defined communications bus that describes the rules for exchanging data frames between devices. Given its limitations mainly in terms of bandwidth and payload size, recently, two main improved communication infrastructures have been proposed. The CAN+ protocol was proposed by Ziermann, *et al.* in 2009 [2], and it exploits the time between transmissions to send additional data. More recently, in 2012, Robert Bosch Gmbh developed the CAN with flexible data-rate protocol (CAN-FD) [3], which brings several advantages over CAN and CAN+, amongst which the most significant being higher bandwidth and larger payload.

978-1-7281-9080-8/20/$31.00

Unfortunately, while the latest advances in the CAN protocol have brought several advantages, today's modern car consists of a heterogeneous environment where the CAN-FD protocol functions coexist alongside the classical CAN. Therefore, this paper proposes a lightweight key generation and data authentication scheme applicable in the case of the classical CAN. The approach builds on Joshua Guttman's authentication tests, as formulated in [4]. The main advantage of this approach is that it only requires a single broadcast message in order to initiate the generation process of a new cryptographic key for a group of Electronic Control Units (ECUs). Experimental results consisting of a laboratory-scale system demonstrate the effectiveness of the developed methodology.

The remainder of this paper is structured as follows. Section 2 summarized the related work relevant to the proposed protocol. Continuing with Section 3, the proposed key management protocol is in depth described. Following with a discussion on the security properties of the proposed solution in Section 4. The experimental assessments conducted and the results obtained are presented in Section 5, and finally, Section 6 concludes the paper.

II. Related Work

The field of key distribution within in-vehicle communication systems has received significant attention. Several techniques have been developed in order to secure CAN communications. Starting with the work of Herrewege, *et al.* [5], it was shown that, while the CAN bus has significant limitations in terms of payload and bandwidth, data authentication is still achievable via Message Authentication Codes (MACs), and more specifically the HMAC standard, alongside a counter in order to ensure freshness and resistance against replay attacks.

By leveraging the breakthrough in the field of sensor networks, namely the Timed Efficient Stream Loss-tolerant Authentication protocol (TESLA) [6], B. Groza, *et al.* [7], developed an approach specially tailored to the CAN protocol. The approach integrated symmetric cryptographic functions (i.e., MAC) for data authentication and for releasing new session keys. Similarly to TESLA, keys are chained and released after the authenticated frames.

Next, we mention the more recent work of Radu and Garcia [8], where the LeiA CAN authentication protocol was presented. LeiA is designed to be backward compatible with existing vehicle's CAN infrastructure and it authenticates each CAN frame by adding one additional frame (and new CAN identifier) to each CAN frame. The newly added frame carries the authentication tag and leverages a 64-bit truncated MAC.

By taking advantage of the CAN-FD protocol, Woo, *et al.* [9], developed a secure communication protocol and security architecture for CAN-FD-enabled in-vehicle systems. The approach involves data encryption (via symmetric cryptography) and authentication (via MAC computation). However, the additional encryption introduces significant overhead on the CAN, which is mitigated by the assumption of CAN-FD as underlying communication infrastructure.

Wang and Liu [10] acknowledged the importance of securing gateway Electronic Control Units (ECUs), which provide various communication interfaces, including the support for Vehicle to Internet communications. Wang's approach is similar to the one proposed in this work, since it focuses on mitigating common network attacks frequently seen on the CAN bus, by developing a security protocol at application level. Furthermore, Wang and Liu provide useful threat analysis and attack models based on ISO 13335 Guidelines for the Management of IT Security for the automotive networks. In contrast to their approach, this paper documents an even simpler key distribution scheme based on on a single message that can be deployed via regular software updates.

III. Developed Approach

By taking advantage of the native properties of the the Controller Area Network (CAN) protocol, such as, its broadcast communication pattern, and its multiple master/slave design, our approach presents a specially tailored, simple, yet effective, key distribution protocol for CAN environments. The *LOKI* protocol intends not to distribute cryptographic keys in the classic sens, but to update session or group key, without the need of the actual transmission of the key. This is done by leveraging a set of bootstrapped cryptographic primitives, and by taking advantage of a reverse hash chain traversal algorithm, namely the *Speed2Pebbling* algorithm. This implies that ECUs are grouped based on their relationship with each other, and that each group has a master ECU that is responsible for two main operations: (i) initiating the generation of a new session key; and (ii) responding to key synchronization requests.

Each group is bootstrapped with a master key denoted by K_g^M, where g is the group identifier, and M denotes the master of the respective group. The master key is a long-term cryptographic key installed during factory setup, and it is presumed to be hardware-protected against changes (e.g., via a Trusted Platform Module – TPM). Similarly, each participant in a group, is bootstrapped with the starting value of a hash chain, and a set of intermediate hash chain values issued by the trusted authority in the protected zone. For each group g, the first value in the chain is considered to be the first session key, denoted as k_g^i, with i as a session identifier. As described in the following section, a set of intermediate points in the chain are used to travel in reverse the chain. This operation servers as a key derivation function (KDF), based on which the session or group key are generated.

There are several advantages of using *Speed2Pebbling* to reverse traverse a given chain in order to generate new keys. First, as described in the following section, the algorithm is limited to a maximum of b hash computations per round. Second, its complexity in terms of storage is limited to $O(b)$, with b being the number of intermediate hashes stored in order to reverse traverse the chain. For example, for a hash chain of length 2^5, only 5 intermediate hashes are stored.

A. Derivation of Cryptographic Keys

Hash chains were designed and introduced by Leslie Lamport in [11] as a technique to authenticate users to a server over an insecure network communication channel. Hash chains proved their usefulness not only in user authentication, but also in lightweight, environment restricted, security protocols. Based on one-way hash functions, hash chains inherit their properties directly, such as: first and second preimage resistance, and collision resistance to attacks. In Lamport's protocol, a hash chain of length n is defined as $f^n(x)$, where x is a random secret seed used to initialized the chain. To compute the hash chain, one must apply n hash operations on x to obtain $f^n(x)$. For example, for length $n = 1000$ hash chain, one it is required to compute 1000 hash operations on the seed x (i.e., $f^{1000}(x)$) to obtain the first value in the chain.

In order to authenticate a user to a server in the schema developed by Lamport, the server is required to store the first value of a hash chain of length n, meaning $f^n(x)$. The user authenticates himself to the server by providing $f^{n-i}(x)$. To verify the user's authenticity, the server must compute i hash operations on $f^{n-i}(x)$ and compare the output to the stored value $f^n(x)$. If the values match, the user is authenticated.

Starting with the work Jakobsson [12], the problem of forward hash chain traversal took a whole new approach. By leveraging b intermediate points in the chain, called pebbles, Jakobsson's approach was able to reduce the number of hash computations required for a given value in the chain, to $O(log^2 n)$. By extending Jakobsson's algorithm, Schoenmaker [13] developed a binary pebbling algorithm capable of traversing in reverse a hash chain of length $n = 2^b$. This was done by storing only maximum b intermediate points, with the maximum $\frac{b}{2}$ hash computations per algorithm round. In this work we take advantage of Schoenmaker's *Speed2Pebbling* algorithm to build a lightweight key distribution protocol over the CAN bus network.

Let $P_b(x)$ be an algorithm that outputs in reverse the hash chain $f_b^n(x)$ using in total $b2^{b-1}$ hash operations, where n is the chain length and b the number of pebbles stored by $P_b(x)$. In the present context, a pebble represents a hash value stored with the purpose of helping $P_b(x)$ traverse in reverse the hash chain $f_b^n(x)$. In each round r, a pebble p_b stored by $P_b(x)$ can have one of the following states:

- *Idle*: if it's in a round r in the interval $[1, 2^{b-1})$.
- *Hashing*: if it's in a round r in the interval $(2^{b-1}, 2^b)$.
- *Redundant*: if it's in a round r in the interval $(2^b, 2^{b+1})$.

Throughout rounds $[1, 2^{b-1})$, $P_b(x)$ produces no output at all, only in the rounds $(2^{b-1}, 2^b)$ hashes are outputted. If a pebble p_b becomes redundant, it is replaced by the pebbles $p_0, ..., p_{b-1}$ in the course of its last $2^b - 1$ output rounds. In order to maintain and determine the state of the pebbles that should run in round r, the binary representation of a counter $c = 2^{b+1} - r$ is used. Furthermore, c is used to count the remaining number of rounds r. As for storage, an array z of length b is used to store the pebbles. Next, the two stages for $P_b(x)$ are described:

- *Initial stage*: During its initial stage, $P_b(x)$ generates the $f_b^n(x)$ chain, and stores b pebbles in z. During this stage, a number of 2^b hash operations are computed.
- *Output stage*: During its output stage, in each round r, a hash value is outputted. In each round r, several pebbles p_b can run in parallel by taking turns to execute and update their position in the chain.

Both stages of the algorithm are briefly described in the following. For a more in depth, step by step analysis of the method, it is recommended that the reader consults [13].

Algorithm 1: Initial stage

Input: s Secret seed
b Number of pebbles
z Array used to store the pebbles
Output: z
Function `Initialization(`b, z`)`**:**
  $q \leftarrow b$
  $y \leftarrow f(s)$
  $i \leftarrow 2^b$
  while $i \geq 1$ **do**
    if $i = 1 << q$ **then**
      z.`Append(`y`)`
      $q \leftarrow q - 1$
    $y \leftarrow f(y)$
    $i \leftarrow i - 1$
  return z
End Function

Algorithm 1 describes the steps to create the hash chain, and the placement of b pebbles in array z. On the other hand, Algorithm 2 describes the execution of the *Speed2Pebbling* algorithm in a round r. First, the output h is updated. Then, using the `HasZeros()` function call, which checks the counter c for trailing 0 bits, it is determined if there are redundant pebbles that require update. If true, the redundant pebbles are updated. Afterwards, the state of the currently running pebble is determined and updated using the `HasOnes()` function. Finally, the `GetNextPebble()` function is called, traversing the binary representation of c and returning the next hashing pebble, which is identified by a 1 bit in c.

Algorithm 2: Output stage for round r

Input: z Array used to store the pebbles
r Current round of the algorithm
Output: h Hash value outputted in round r
Function `Round(`z, r`)`**:**
  $h \leftarrow z[0]$ Hash value that will be returned
  $c \leftarrow r$ Set c to the current r value
  $i \leftarrow 0$ Initialize i index for redundant pebble
  while *HasZeros(c)* **do**
    $z[i] \leftarrow z[i+1]$ Update redundant i
    $i = i + 1$
    $c = c >> 1$
  $i = i + 1$
  $c = c >> 1$
  $q = i - 1$
  while *HasOnes(c)* **do**
    $z[q] \leftarrow f(z[i])$ Update current q pebble
    if $q \neq 0$ **then**
      If q is redundant
      $z[q] \leftarrow f(z[q])$ Update it
    $c \leftarrow$ `GetNextPebble(`c`)`
    $q \leftarrow i$
  return h
End Function

Algorithm 3: Generating a new group key

Input: z Array used to store the pebbles
r Current round of the algorithm
Output: k_g^{i+1} New group key
Function `GenerateGroupKey(`z, r`)`**:**
  $k_g^{i+1} \leftarrow$ `Round(`z, r`)`
  `Store(`k_g^{i+1}`)`
  $t \leftarrow$ `GetCurrentTime()`
  $input \leftarrow k_g^i || t || \texttt{CID} || g$
  $mac_g^M \leftarrow \texttt{MAC}(input, K_g^M)$
  $mac_{g_{trunk}}^M \leftarrow \texttt{Trunk}(mac_g^M, 64)$
  `Send(`$mac_{g_{trunk}}^M$`)`
End Function

B. Generating Fresh Cryptographic Keys

The method of generating a fresh, new group cryptographic key, named in the following as *group key*, consists of a single message that is broadcasted by the group's master ECU, further denoted as ECU_g^M. The following message is broadcasted by ECU_g^M, to all group members denoted by $ECU_g^{\forall id}$ (where id is the ECU's unique identifier):

$$ECU_g^M \rightarrow ECU_g^{\forall id} : \texttt{MAC}(k_g^i || t || \texttt{CID} || g, K_g^M).$$

In the previous equation, a MAC function is computed by ECU_g^M over the current session key k_g^i, a timestamp t, concatenated with the message CAN identifier (`CID`) of

Algorithm 4: Verification of a new group key

Input: $mac^M_{g_{trunk}}$ MAC received from master M of group g
z List of pebbles g
r Current algorithm round g

Output: k_g^{i+1} New group key
Function `VerifyGroupKey` $(mac^M_{g_{trunk}}, z, r)$ **:**
  $t \leftarrow$ `GetCurrentTime()`
  $curr \leftarrow 1$
  while $curr \leq R$ **do**
    $mac \leftarrow \texttt{MAC}(k_g^i||t||CID||g, K_g^M)$
    $mac_{trunk} \leftarrow \texttt{Trunk}(mac, 64)$
    if $mac^M_{g_{trunk}} \neq mac_{trunk}$ **then**
      $curr \leftarrow curr + 1$
      $t = t + 1$
    else
      $k_g^{i+1} \leftarrow \texttt{Round}(z, r)$
      `Store`(k_g^{i+1})
      `return`
  if $curr = R$ **then**
    `Synchronize()`
End Function

the current frame, and the group identifier g, by leveraging the group's master key K_g^M. This procedure is described in Algorithm 3. First of all, ECU_g^M generates a new session key by executing Algorithm 2. Namely, the hash chain is traversed in reverse and the next hash in the chain is used to update the previous session key k_g^i. To compute the previously mentioned operation, the session key k_g^i is concatenated with the global timestamp, with the frame `CID` and the group identifier g. In order to fit this construction in a single CAN frame, according to the recommendations set by AUTOSAR [14] and NIST [15], the MAC tag is truncated to 64 bits.

By broadcasting this cryptographic construction, the master ECU_g^M initiates, for each group member, the procedure for generating the next group key.

In order to successfully verify the authenticity and integrity of a received MAC, the synchronization of timestamps is critical. Given that the timestamp is not sent explicitly by ECU_g^M, but it is obtained by ECUs via the time synchronization protocol, the resolution of the timestamp needs to be in the level of seconds. However, we observe that even so, a slight delay in the verification can yield a difference in timestamps of several seconds. Thus, the verification procedure, if it fails the first time, gradually, a set of retries are executed with the value of the timestamp t incremented by one on each attempt. In this process, R denotes the maximum number of attempts that a verifier ECU should go through. The optimal value for R should be determined at system design time by the Original Equipment Manufacturer (OEM). This procedure is described in Algorithm 4.

If the received MAC is successfully authenticated, the verifier ECU proceeds to compute the next group key in the same fashion as the master, by executing `Round()` algorithm to obtain the next hash from the chain.

It is to be noted that this process can lead to miss synchronization. First, in terms of timestamps, and second, in terms of pebbles. To address these issues, we presume that the time protocol that runs in the network conforms to the AUTOSAR standard, thus, a time synchronization protocol is in place. Conversely, for the pebbles, a synchronization procedure must take place between the ECU slave, ECU_g^S and the master ECU_g^M. This procedure follows a challenge-response communication pattern that is described in the next section.

In terms of security construction, this scheme adheres to Joshua Guttman's authentication tests, as formulated in [4]. More specifically, the scheme adheres to Guttman's unsolicited authentication test, which includes a single message. While weaker in terms of authentication tests, the recipient can verify the freshness and the authenticity of the message.

C. Key Synchronization Protocol

A possible scenario for the proposed protocol is that a given ECU_g^S may miss, or not receive the MAC tag sent by ECU_g^M that triggers the key generation protocol. Thus, a synchronization protocol must exists for ECU_g^S to request the stored pebbles from ECU_g^M.

The synchronization procedure consist of the following steps:

$$Challenge : ECU_g^S \rightarrow ECU_g^M : N^S|| \\ \texttt{MAC}(N^S||t||\texttt{CID}||g, K_g^M),$$

$$Response : ECU_g^M \rightarrow ECU_g^S : \{P_b^M\}_{K_g^M}|| \\ \texttt{MAC}(\{P_b^M\}_{K_g^M}||N^S + 1||t||\texttt{CID}||g, K_g^M).$$

The synchronization protocol follows a challenge-response pattern, in accordance with Joshua Guttman's authentication tests [4], where ECU_g^S challenges ECU_g^M, in the same time while requesting the master's pebbles. In the equations above, N^S denotes a nonce generated by the ECU_g^S. In the challenge, along with the nonce N^S, a MAC tag is sent, which is computed over the nonce, the current timestamp t, the CAN frame identifier CID, and the group identifier g, using the master key K_g^M.

Upon receiving the challenge, ECU_g^M proceeds with the MAC verification. If successful, it responds with a series of messages that contain the encrypted pebbles P_b^M with the master key K_g^M, denoted by the operation $\{\}$. Besides this, every encrypted pebble is transmitted together with a MAC tag computed over the encrypted pebble $\{P_b^M\}_{K_g^M}$, the nonce N^S incremented by one, a timestamp t, the CAN identifier CID, and, the group identifier g, by leveraging the same master key K_g^M. Once again, this construction is in accordance with Joshua Guttman's authentication tests [4].

Algorithm 5: Key Synchronization

Input: K_g^M Master key of group g

Output: k_g^{i+1} New group key

Function `Synchronize()`:

$mac \leftarrow \texttt{MAC}(N^S||t||\texttt{CID}||g, K_g^M)$

$mac_{trunk} \leftarrow \texttt{Trunk}(mac, 64)$

$\texttt{Send}(< N^S, mac_{trunk} >)$

$< \{P_b^M\}_{K_g^M}, mac_{g_{trunk}}^M > \leftarrow \texttt{Receive()}$

$mac' \leftarrow$
$\texttt{MAC}(\{P_b^M\}_{K_g^M}||N^S+1||t||CID||g, K_g^M)$

$mac'_{trunk} \leftarrow \texttt{Trunk}(mac', 64)$

if $mac'_{trunk} = mac_{g_{trunk}}^M$ **then**

$\texttt{UpdatePebble}(P_b, \{P_b^M\}_{K_g^M}, K_g^M)$

End Function

Here, the number of CAN messages required in order to synchronize a slave ECU_g^S with a master ECU_g^M depends on the length of the hash function used in the computation of the hash chain, and the number of pebbles used in reverse traversing the chain.

Upon receiving the requested pebbles, ECU_g^S checks the integrity of the MAC tag before it proceeds to update the stored pebbles. If the MAC tag verification is successful, then ECU_g^S decrypts pebble P_b^M and updates the stored pebbles. The key synchronization is summarized in Algorithm 5.

D. Slave Authentication

It should be noted that the developed scheme offers several notable advantages, namely: reduced number of CAN messages required to generate a new group key; a reduced impact on the CAN bus load; and, a lightweight structure where slaves do not respond to the master's requests. Nevertheless, these advantages entail that slaves are authenticated later against their knowledge on the new group key.

To verify that the generation of the new group key k_g^{i+1} succeeded across all the members of the group g, ECU_g^M verifies the authenticity of incoming CAN messages from each ECU_g^S in a given time window. If ECU_g^M is able to verify the signed messages with the new group key, then, the key distribution protocol can be considered successfully completed.

E. Security Analysis

There are several security properties defined in the literature [16], [17], that a group key management schema should guarantee. Thus, such an approach should encompass the following:

1) *Forward secrecy*: guarantees that even if a set of previously released set of keys are known by an adversary, he/she is unable to discover any future group keys.
2) *Backwards secrecy*: similar to *forward secrecy*, guarantees that even if a set of previously released set of keys are known by an adversary, he/she cannot discover previously released group keys.
3) *Group key secrecy*: guarantees independence between groups, by leveraging different cryptographic keys shared among the participants, while in the same time making it computationally infeasible for an adversary to compromise a group key.

The first two properties, *forward* and *backward secrecy*, are satisfied by *LOKI* due to the properties inherited from hash chains. Here, it is computationally infeasible for a adversary to try to compute a new session key, after compromising a previously released key, due to the nature and guarantees offered by hash functions. Furthermore, the secret seed from which the hash chain was generated, is not known by the communication participants, nor it can be known by the adversary, but only by the OEM trusted authority.

The *group key secrecy* property is guaranteed by the possibility of using a set of different cryptographic primitives for each group. This means that each group can be bootstrapped with a different master key K_g^M, and with a different hash chain. Consequently, even in the worse case scenario where a group is compromised by an adversary, this event will not affect other existing groups.

Lastly, we note that *LOKI* complies to the existing Secure On Board Communication Standard developed by AUTOSAR [14]. First, it complies in terms of key length usage, which are at least 128-bit in length. Second, freshness is ensured by timestamps, which are obtained via the time synchronization protocol. Last but not least, 64-bit length truncated MAC authentication tags are used to sign the sent CAN messages.

IV. Experimental Assessment

To demonstrate the applicability of the proposed key management protocol, its impact has been measured in a close to reality simulated CAN bus, in terms of bus load for different bus baud rates. First of all, the CAN bus network was designed. Then, for each chosen baud rate, a base line was measured. Finally, the proposed protocol was integrated and ran over the same CAN bus for each baud rate chosen and the results obtained were compared to the baseline ones.

A. Implementation Details

In order to simulate a close to reality CAN bus network, the simulation environment CANoe/CANalyzer [18] provided by Vector Informatik was used. In terms of implementation, the proposed protocol was implemented using the C and C++ programming languages. For cryptographic primitives, the Win32 certified API was used. To run the protocol in CANoe, a Windows Dynamic Linked Library (DLL) that implements the CANoe programming interface was developed. By opting to implement the CANoe programming interface as such, the C/C++ functions have been made available to the build-in CANoe scripting language CAPL, which was used to program the simulated ECUs behaviour. Finally, regarding the protocol parameters, the implementation used: a 256 bit-length master key, a 128 bit-length secret seed, SHA-1 to compute

TABLE I
MINIMUM, MAXIMUM AND AVERAGE BUSLOAD FOR CAN BUS RUNNING AT 100 KBPS BAUD RATE.

	Min. Busload	Max. Busload	Avg. Busload
LOKI and SecOC	53.06%	55.80%	55.20%
LOKI	30.87%	32.89%	32.31%
Normal	30.87%	32.54%	32.25%

TABLE II
MINIMUM, MAXIMUM AND AVERAGE BUSLOAD FOR CAN BUS RUNNING AT 125 KBPS BAUD RATE.

	Min. Busload	Max. Busload	Avg. Busload
LOKI and SecOC	42.45%	44.64%	44.15%
LOKI	24.70%	26.31%	25.86%
Normal	24.70%	26.03%	25.80%

TABLE III
MINIMUM, MAXIMUM AND AVERAGE BUSLOAD FOR CAN BUS RUNNING AT 250 KBPS BAUD RATE.

	Min. Busload	Max. Busload	Avg. Busload
LOKI and SecOC	21.23%	22.32%	22.08.%
LOKI	12.35%	13.15%	12.93%
Normal	11.67%	12.10%	12.03%

the hash chain, and, 64 bit truncated HMAC using SHA-1 to compute the message authentication tags.

In terms of network design, a single simulated CAN bus network was considered. On this network, the Global Time protocol was activated in order to provide the freshness values for the protocol. Overall, three groups of ECUs have been designed such that each group consisted of one group master and three slave ECUs. The build-in Generator feature from CANoe was used to periodically generate burst of CAN frames to influence the bus load.

B. Scenario

By following the implementation and design details presented in the previous section, the experimental results were obtained in the following manner. First, a set of bus baud rates were chosen: 100, 125 and 250 Kbps. For each baud rate, a busload baseline was measured in the absence of the *LOKI* protocol. Afterwards, the *LOKI* protocol was enabled. Every group master ECU_g^M periodically, at a predefined time, initiated the generation of a new group key. Each slave $ECU_g^{\forall id}$ updated accordingly its group key by following the protocol. Subsequently, according to AUTOSAR's SecOC (Secure On Board Communication) standard, each slave $ECU_g^{\forall id}$ periodically sent authenticated traffic using a 64-bit truncated MAC. Lastly, in every measurement, the Global Time protocol was executed.

C. Results

For each baud rate, 100 Kbps, 125 Kbps, 250 Kbps, the minimum, maximum and average busload were measured in three different contexts. The first context refers to the CAN bus busload in terms of normal, security protocol-free setup, which also represents the baseline for the following experiments. This is denoted by the term `Normal` in the followings. The second context consists of running the simulation with the *LOKI* protocol, and it is identified by the term `LOKI` (i.e., Lightweight Cryptographic Key Management Protocol). Finally, besides the key management protocol, an additional measurement was conducted, where the CAN frames sent by ECUs were sent in parallel with truncated MAC tags, as specified by AUTOSAR SecOC.

Tables I, II, III summarize the results of the conducted experiments. As shown here, the impact of the developed scheme on the overall traffic is reduced. Namely, the additional CAN bus load rounds in the average at maximum 1%. This is owed to the additional CAN messages, which have been added to the system are part of the developed security scheme. The additional increase of the CAN bus load in the case of activating the SecOC is owed to the fact that, after the execution of the proposed key distribution scheme, each CAN frame is authenticated by a truncated MAC. More specifically, according to SecOC, in the present experiment, for each CAN frame an additional CAN frame was sent, which contained the authentication tag.

V. CONCLUSIONS

In this work, we designed and implemented the *LOKI* protocol. *LOKI* provides a lightweight key management protocol, which is backwards compatible with the Control Area Networks protocol. By taking advantage of a reverse hash chain algorithm in order to generate new session or group keys, *LOKI* minimizes the overhead on the communication bus, while maintaining the normal operation of the in-vehicle communication systems. The measurements prove the viability of key management protocol proposed in the current work.

As future work, our intent is to implement the *LOKI* protocol in a real CAN test-bed, that contains similar hardware with the hardware found in Electronic Control Units (ECUs) and measure its performance more accurately. Furthermore, we observed that certain improvements can be made in order to reduce the overhead brought by the pebble synchronization protocol. Here, we intend to develop a method of synchronization that eliminates the need for the actual transmission of the pebbles. To this end, if a slave ECU is able to communicate the state of it's hash chain to a master, in response, the master could determine in relation with it's chain state the number of rounds required by the slave to be computed in order to synchronize itself with the group. Consequently, this would fully eliminate the need for transmission of any information regarding the key management protocol, and, it would further reduce the bus overhead.

REFERENCES

[1] ISO, "ISO 11898-1:2003 - Road vehicles - Controller area network (CAN) - Part 1: Data link layer and physical signalling," *International Organization for Standardization*, 2003.

[2] T. Ziermann, S. Wildermann, and J. Teich, "CAN+: A new backward-compatible Controller Area Network (CAN) protocol with up to 16 higher data rates.," in *2009 Design, Automation Test in Europe Conference Exhibition*, Apr. 2009, pp. 1088–1093. DOI: 10.1109/DATE.2009.5090826.

[3] Robert Bosch Gmbh, "Can with flexible data-rate," *Vector CANtech, Inc., MI, USA, Specification Version 1.0*, 2012.

[4] J. D. Guttman, "Security protocol design via authentication tests," in *Proceedings 15th IEEE Computer Security Foundations Workshop. CSFW-15*, 2002, pp. 92–103.

[5] A. Van Herrewege, D. Singelee, and I. Verbauwhede, "CANAuth - A Simple, Backward Compatible Broadcast Authentication Protocol for CAN bus," in *ECRYPT Workshop on Lightweight Cryptography 2011*, ser. ECRYPT '11, 2011, pp. 1–7.

[6] A. Perrig, R. Canetti, J. D. Tygar, and Dawn Song, "Efficient authentication and signing of multicast streams over lossy channels," in *Proceeding 2000 IEEE Symposium on Security and Privacy. S P 2000*, May 2000, pp. 56–73. DOI: 10.1109/SECPRI.2000.848446.

[7] B. Groza and S. Murvay, "Efficient Protocols for Secure Broadcast in Controller Area Networks," *IEEE Transactions on Industrial Informatics*, vol. 9, no. 4, pp. 2034–2042, Nov. 2013, ISSN: 1941-0050. DOI: 10.1109/TII.2013.2239301.

[8] A.-I. Radu and F. D. Garcia, "LeiA: A Lightweight Authentication Protocol for CAN," in *Computer Security – ESORICS 2016*, I. Askoxylakis, S. Ioannidis, S. Katsikas, and C. Meadows, Eds., Cham: Springer International Publishing, 2016, pp. 283–300, ISBN: 978-3-319-45741-3.

[9] 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, no. 8, pp. 2248–2261, Aug. 2016, ISSN: 1558-0016. DOI: 10.1109/TITS.2016.2519464.

[10] L. Wang and X. Liu, "NOTSA: Novel OBU With Three-Level Security Architecture for Internet of Vehicles," *IEEE Internet of Things Journal*, vol. 5, no. 5, pp. 3548–3558, Oct. 2018, ISSN: 2372-2541. DOI: 10.1109/JIOT.2018.2800281.

[11] L. Lamport, "Password authentication with insecure communication," *Commun. ACM*, vol. 24, no. 11, pp. 770–772, Nov. 1981, ISSN: 0001-0782. DOI: 10.1145/358790.358797. [Online]. Available: https://doi.org/10.1145/358790.358797.

[12] M. Jakobsson, *Fractal hash sequence representation and traversal*, Cryptology ePrint Archive, Report 2002/001, https://eprint.iacr.org/2002/001, 2002.

[13] B. Schoenmakers, *Explicit optimal binary pebbling for one-way hash chain reversal*, Cryptology ePrint Archive, Report 2014/329, https://eprint.iacr.org/2014/329, 2014.

[14] AUTOSAR, "Specification of Secure Onboard Communication AUTOSAR CP Release 4.3.1," *AUTOSAR*, 2017.

[15] Q. H. Dang, *Recommendation for Applications Using Approved Hash Algorithms, Special Publication (NIST SP) - 800-107 Rev 1*, 2012. [Online]. Available: https://www.nist.gov/publications/recommendation-applications-using-approved-hash-algorithms.

[16] A. Penrig, D. Song, and D. Tygar, "Elk, a new protocol for efficient large-group key distribution," in *Proceedings 2001 IEEE Symposium on Security and Privacy. S P 2001*, 2001, pp. 247–262.

[17] M. Steiner, G. Tsudik, and M. Waidner, "Cliques: A new approach to group key agreement," in *Proceedings. 18th International Conference on Distributed Computing Systems (Cat. No.98CB36183)*, 1998, pp. 380–387.

[18] Vector Informatik, *CANoe v 12.0: Testing ECUs and Networks with CANoe*, Last access: January 6th, 2020, 2020. [Online]. Available: https://www.vector.com/int/en/products/products-a-z/software/canoe/.

ACKNOWLEDGMENT

This work was funded by the European Union's Horizon 2020 Research and Innovation Programme through DIAS project (https://diasproject.com/) under Grant Agreement No. 814951.