Fast and Efficient Group Key Exchange in Controller Area Networks (CAN)

Adrian Musuroi, Bogdan Groza, Lucian Popa and Pal-Stefan Murvay

Abstract—**The security of vehicle communication buses and electronic control units has received much attention in the recent years. However, while essential for practical deployments, the problem of securely exchanging cryptographic keys between electronic control units on the CAN bus received little attention so far. In this work, we evaluate group extensions of a regular key exchange protocol, i.e., the elliptic curve version of the Diffie-Hellman protocol, by using both a standardized NIST elliptic curve as well as the faster, more recently proposed Four$\mathbb{Q}$ curve. We deploy protocol implementations and determine crisp performance bounds on real-world automotive-grade platforms with Infineon and ARM cores. For an up-to-date analysis, we use both CAN and its more recent extension CAN-FD as communication layers. Roughly, the computational runtime of the key exchange protocol scales logarithmically or linearly with the number of nodes, depending on the protocol version. The computational time proves to be more critical than bandwidth due to the more demanding elliptic curve operations.**

Index Terms—**CAN bus, authentication, group key exchange, elliptic curve, Four$\mathbb{Q}$**

I. Introduction and motivation

While largely used as a communication layer by most Electronic Control Units (ECU) inside vehicles, the CAN bus and its newer embodiment with flexible data-rates CAN-FD, have no intrinsic security. The consequences of this have been largely proved in works such as [1], [2], [3] and in some more recent papers exploiting remote vulnerabilities [4], [5]. A recent survey on existing challenges for wired and wireless vehicular buses can be found in [6]. The effects may range from small nuisances, e.g., degradation of certain vehicle functions, up to major problems that may lead to life-critical malfunctions of the vehicle. Consequently, there are no doubts that the security of in-vehicle buses is a problem that needs special attention.

The CAN bus was designed in the 80's by BOSCH as a two wire differential bus, that is cost effective and has high reliability. Figure 1 illustrates the topology of the CAN bus. Each ECU connects to the differential lines of the bus CAN-H and CAN-L. CAN follows a serial multi-master communication model with broadcast transmission. CAN resolves bus contention using an ID-oriented arbitration. That is, each message contains an ID field that determines which is the node allowed to continue the transmission. An 11 bit ID accompanies each standard data frame, while extended frames contain a 29 bit ID field for the same purpose, as depicted in Figure 2 which presents the structure of an extended data frame. While usually the ID field denotes the frame content, there are multiple implementations of the CAN upper layers (e.g. SAE J1939, ISO-TP) that use the ID field to indicate the sender. Even if the upper layers of the employed protocol do not specifically designate the ID as an indicator of the message sender it can still be used to identify the transmitter since most frame types can only be sent by a particular ECU. Therefore, it is expected that data-frames with the same ID originate from the same sender. Consequently, parts of the ID field can be used to encode the node addresses.

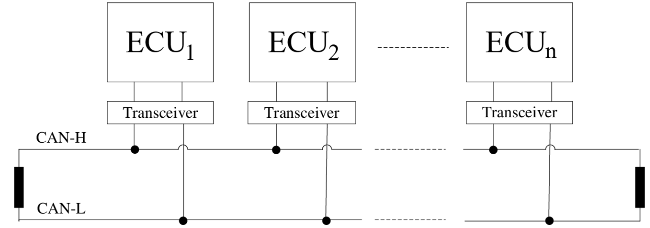


Fig. 1. Depiction of the CAN Bus

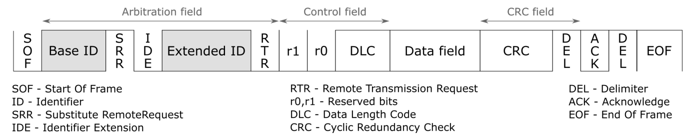


Fig. 2. The structure of an extended CAN frame

Architecture and security requirements. From a networking perspective, ECUs inside a car are usually grouped into sub-networks dedicated to specific functionalities, i.e., body, chassis, powertrain, etc. We suggest this in the conceptual CAN bus architecture from Figure 3 which is inspired from real-world in-vehicle networks such as [7], [8]. The intercommunication between ECUs on distinct sub-buses is mediated by a gateway ECU that is responsible for traffic redirection. In response to the recent attacks, numerous research proposals appeared for securing the CAN bus and the industry also reacted with current standards for secure on-board communication, i.e., AUTOSAR [9]. This standard in particular requires the presence of authentication elements and freshness parameters in each frame. The computation of the authentication element

Copyright (c) 2021 IEEE. Personal use of this material is permitted. However, permission to use this material for any other purposes must be obtained from the IEEE by sending a request to pubs-permissions@ieee.org.

Adrian Musuroi, Bogdan Groza, Lucian Popa and Pal-Stefan Murvay are with the Faculty of Automatics and Computers, Politehnica University of Timisoara, Romania, Email: {adrian.musuroi, bogdan.groza, lucian.popa, pal-stefan.murvay}@aut.upt.ro

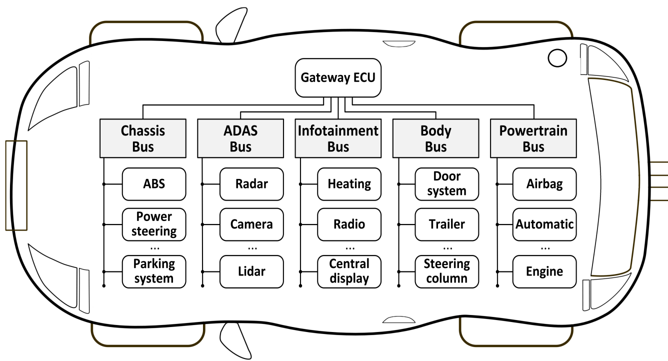


Fig. 3. Conceptual CAN architecture

in each frame is done via the CMAC algorithm based on AES-128. The output of the CMAC is further truncated to 24-28 bits depending on the profile [9].

Existing challenges. All of the proposed cryptographic methods to protect the CAN bus, including the aforementioned CMAC-AES from [9], require a secret key to be shared between ECUs for encryption/authentication purposes, i.e., a group key. For these purposes, group key exchange protocols are needed. This is a well explored topic in the domain of distributed systems, e.g., [10], [11], [12]. In the case of the CAN bus however, there was little effort so far in designing or evaluating group key exchange protocols. Existing AUTOSAR standards, e.g., [13], [14], are specifying function templates for key exchanges between pairs of ECUs, but obviously these need to be scaled up to multiple ECUs on the bus. Some early works that addressed group key exchange on CAN buses are [15], [16] and [17] which securely exchange cryptographic keys based on the wired-AND behavior of the CAN physical layer. But using the physical layer may open road to probing attacks, as demonstrated in [18]. So far, for practice, cryptographic protocols seemed to be the only secure alternative. More recently, the authors in [19] and a few other works (discussed in the related work section) explored several cryptographic protocol versions for exchanging group keys. But all these versions are more computationally demanding compared to the protocols evaluated in this work.

Contribution and relation to existing works. The theoretical foundations of group key exchange have been carefully investigated by cryptographers and security engineers for at least three decades, e.g., [10]. On the practical side of in-vehicle networks, only recently, the authors in [16] explored a group key exchange protocol that uses a tree to group ECUs connected to the CAN bus. The idea of grouping nodes in a tree to perform a group key exchange is not new and can be traced back to earlier works starting from the 90's, e.g., [20], while subsequent works brought several improvements by extending trees to graphs [21] and embedding the Diffie-Hellman key exchange paradigm in the key tree, i.e., [11] and [12]. However, the group key agreement for CAN proposed in [16] uses the physical bus which comes with a disadvantage: the key exchange can be performed only for nodes that are connected to the same bus, i.e., the physical layer key exchange cannot pass through gateways that connect different sub-buses. This is not a limitation in case of regular cryptographic key exchange protocols, e.g., Diffie-Hellman based, which do not require nodes to be connected to the same physical bus. Grouping nodes under a key exchange tree also allows to harness the speed of parallel protocol executions on multiple ECUs and gives more flexibility to the group key management. In our proposal we employ a Security Orchestrator ECU (SoECU) that is responsible for the group key agreement. This is in line with early approaches in group key exchange such as [22] where a group key controller is responsible for coordinating the group key agreement. The gateway ECU suggested by us in Figure 3 is the most suitable ECU to be in charge with the entire group key exchange orchestration, i.e., the role of the Security Orchestrator ECU (SoECU) that will coordinate the nodes to reach a common secret key on each of the sub-buses. Our protocol proposals differ from previous approaches in the way we orchestrate messages on the CAN bus, how we allocate protocol frames over CAN IDs and the AUTOSAR compatibility of our application stack. Also, different to related works, here we consider the fast, more recently proposed, Four$\mathbb{Q}$ elliptic curve [23]. This curve can be efficiently used for implementing cryptographic key exchange protocols for groups of ECUs connected via CAN. We test the performance of the schemes by using automotive-grade platforms. By using the Four$\mathbb{Q}$ elliptic curve, the computations are several times faster than on a regular 256-bit NIST elliptic curve. This allows us to perform a key exchange between two automotive-grade controllers in a few dozen milliseconds and scales up logarithmically or linearly, depending on the protocol version, to an arbitrary number of controllers. Such runtimes are affordable during various routines, e.g., at vehicle start-up. Briefly, the contributions of our work can be summarized as follows:

1) we accommodate and compare several group key exchange protocols on the CAN bus, designing a Security Orchestrator (SoECU) that coordinates ECUs to obtain a common group key,
2) we develop an AUTOSAR compliant implementation for the proposed group key exchange on the CAN bus that is in line with industry requirements,
3) we take benefit of the faster Four$\mathbb{Q}$ elliptic curve which in our experiments leads to a five time decrease in the computational time compared to the commonly recommended NIST P-256 curve,
4) we provide comprehensive experiments on representative automotive-grade controllers and clear performance metrics with respect to computational time and bandwidth.

The rest of the work is organized as follows. Section II surveys over related works. Then in Section III we discuss the basic cryptographic protocols that we use and the group extension that will be further implemented. Section IV holds details on the implementation and Section V the experimental results that we obtain. Finally, Section VI holds the conclusion of our work.

II. Related work

We now discuss previous research on key exchange approaches for automotive networks as well as the integration, implementation and evaluation of elliptic curve libraries on automotive embedded platforms.

Elliptic curve libraries. Several works analyze the memory requirements and time budgets for elliptic curve implementations. The authors in [24] describe a fast, low-power implementation of an elliptic curve library tested using different NIST curves on an ARM Cortex-M0+ CPU. In [25], the authors propose the implementation of elliptic curve pairings on an ARM Cortex-M0+ CPU with hardware extensions emphasizing the timings and memory usage for each operation and proving the practicality of their proposal in embedded applications. Optimizations of the elliptic-curve Diffie-Hellmann key agreement protocol, using Curve25519 as the underlying curve, on 16-bit and 32-bit MSP430 low-power microcontrollers from Texas Instruments are presented in [26] with comprehensive details for code space and execution time performance. The performance of the uECC library (http://kmackay.ca/micro-ecc/), a small open-source cryptographic library with implementations for the elliptic curve digital signature algorithm (ECDSA) and Diffie-Hellman key exchange, is evaluated in [27] with regards to execution timing on an Atmel SAMR21-XPRO board equipped with the ARM Cortex-M0+ CPU. Given the increasing interest in automotive security, there are many proposals for securing in-vehicle communication or integrating secure protocols in existing vehicle systems for verifying software authenticity (i.e. using public key certificates). In the context of the automotive real-time requirements, the computational timing of elliptic curve operations is evaluated by the authors from [28] and [29] on automotive-grade microcontrollers, e.g., Infineon AURIX TC297, using several open-source cryptographic libraries, e.g., wolfSSL (https://github.com/wolfSSL), RELIC[30] or MIRACL (https://github.com/miracl/MIRACL).

Key exchange. One category of approaches proposed for key exchange in automotive networks relies on the behavior of the physical layer. Mueller and Lothspeich are the first to propose using the wired-AND behavior of the CAN physical layer to implement key agreement between two nodes [15]. Their scheme involves concurrent transmission from the two nodes involved in the key exchange from which the legit nodes can extract a common secret. A later extension of this scheme also provides support for group key agreement [16]. A common downside of this approach is that it requires dedicated hardware as it is not supported by standard CAN controllers. However, the mechanism was successfully adapted for the use in achieving key agreement for FlexRay networks without the need of hardware changes [42]. While the basic scheme is vulnerable to physical probing attacks, the work in [18] proposes several efficient solutions for alleviating these threats. The work in [17] provides another mechanism that uses the non-destructive CAN arbitration along with frame delays to implement key exchange on CAN without the need of any additional hardware. The performance analysis was conducted on the Infineon TC297 platform, yielding a computational overhead between $26ms$ and $98ms$ for a two-party key exchange.

Other lines of work have focused on applying various key exchange protocols on CAN. In [33], Woo et al. proposed the use of the AKEP2 [34] protocol which relies exclusively on symmetric primitives. The performance analysis of this mechanism was evaluated using OpenSSL (https://github.com/openssl/openssl) on the Infineon TC275 platform and the computational overhead for establishing a secret key between 20 ECUs clocked at 30MHz was estimated to be under $280ms$. A more recent proposal which also relies only on symmetric cryptography was proposed by Youn et al. in [35]. In this paper, one of the benchmark platforms is Infineon's XC2265N and the results show an execution time of $23ms$ for a key agreement between 15 ECUs. Several protocols based on elliptic curve cryptography (ECC) were also evaluated in other works. While more computationally expensive, these proposals offer more flexibility by removing the overhead of requiring secret keys to be pre-shared between the ECUs. One of the former investigations of using ECC for key agreements in automotive systems was done in [43]. However, the performance evaluation is limited to the analysis of bus load variations when using the proposed mechanism. Multiple versions of identity-based key exchange (IBKE) protocols were explored in [19] and evaluated on Infineon's TC297 platform. These protocols proved to be computationally demanding, e.g., a key exchange between two ECUs was measured to be $191ms$ when evaluating Cao's IBKE. A lightweight scheme which uses PUFs for authentication and implicit certificates for deriving message-based group keys with implementation based on BearSSL (https://bearssl.org/) is proposed in [38]. Here, the benchmark results indicate over $2s$ for authenticating a device and $135ms$ for establishing a single secret key between 30 ECUs (however, the authors suggest using multiple keys, which significantly increases the runtime). In [39], Tatara et al. explored the use of broadcast encryption for updating secret keys. The authors used a Raspberry Pi 3B platform and several libraries, GNU MP (https://gmplib.org/), OpenSSL and ELiPS (https://github.com/ISecOkayamaUniv/ELiPS), for the protocol evaluation. Their experimental results indicate an overhead of $70ms$ for the secret key update operation. Another work that focuses on the use of ECC for key establishment [41] proves the feasibility of the proposed approach by analyzing its application for the real-time requirements of a steer-by-wire system. In this case, the performance was evaluated on a more expensive and high-end platform, i.e., NVIDIA Jetson TX2 which conducts two-party key exchanges in $13ms$.

As a summary for existing proposals on key agreements for the CAN bus, we provide a briefing on the previously mentioned related works in Table I. In the first two columns of the table we depict the employed cryptographic protocols and the software libraries used for implementing them. We note that the first two works from the table, i.e., [15] and [16], employ the physical layer without explicitly using any cryptographic primitive and there are no associated experiments. The next two columns specify the intended bus, i.e., CAN vs. CAN-FD, as well as the hardware platforms that were employed for analyzing the performance. Finally, the last column highlights

TABLE I
SUMMARY OF SUGGESTED APPROACHES FOR KEY AGREEMENT ON CAN/CAN-FD

Work	Cryptography	Libraries	Bus	Platforms	Protocol features
[15]	-	-	CAN	-	Uses the wired-AND behaviour of the physical layer (voltage)
[16]	-	-	CAN	-	Extends [15] to a group key agreement
[17]	ECDH EKE [31] SPEKE [32]	MIRACL	CAN	Infineon TC224 Infineon TC277 Infineon TC297	Physical layer (timing) in the main scheme, extensions with ECC also explored
[33]	AKEP2 [34]	OpenSSL	CAN-FD	TI TMS320C28346 TI TMS320F28335 Infineon TC275	Key distribution/update based on symmetric cryptography (requires pre-shared keys)
[35]	Own	-	CAN	Infineon XC2265N Arduino ATmega328	Key distribution/update based on symmetric cryptography (requires pre-shared keys)
[19]	ECDH Wang IBKE [36] Cao IBKE [37]	MIRACL	CAN-FD	Infineon TC297 SAM V71 XULTRA	Identity-based cryptography to remove certificates, uses pairing-friendly curves
[38]	ECDH	BearSSL	CAN/CAN-FD	Olimex ESP32-EVB Raspberry Pi 3B+	PUFs for authentication, implicit certification, ECDH on curve P-256 and One-Time Pad for key computation
[39]	BGW [40]	GNU MP OpenSSL ELiPS	CAN	Raspberry Pi 3B	Key update with BGW on the pairing-friendly curve BLS12
[41]	ECDH	-	CAN-FD	NVIDIA Jetson TX2	Uses elliptic curve integrated encryption scheme on curve P-192, solution implemented on high performance GPU
Ours	ECDH	FourQlib RELIC	CAN/CAN-FD	Infineon TC224 Infineon TC277 Infineon TC297 SAM V71 XULTRA	High-performance Four$\mathbb{Q}$ curve (NIST's Curve P-256 as reference)

the key aspects in each of the proposals. The performances for each proposal were already mentioned but it may be useful to recap them. Proposals [15], [16] and [17] are based on the physical layer and a direct comparison to the other approaches that use cryptography may be biased. Physical layer approaches are faster, e.g., a key exchange between two nodes can be achieved in well below $1ms$, but they are also vulnerable to new attacks, e.g., probing attacks [18]. Finally, there is no concrete performance evaluation in [15] and [16]. The approaches from [33] and [35] use only symmetric keys, which also makes them more computationally efficient, but they require fixed pre-shared keys which compromises scalability. Finally, the rest of the proposals, i.e., [19], [38] and [39] are all based on asymmetric cryptography, i.e., ECC, and they all seem to be slower than our Four$\mathbb{Q}$-based approach. Only the work in [41] reports results that are faster than ours, i.e., $12.93ms$ compared to $24ms$, but the work is based on a high performance GPU that may be unrealistic for current in-vehicle ecosystems, i.e., their NVIDIA Jetson TX2 runs at 2GHz while our Infineon TC297 controller (a common high-end in-vehicle controller) is only at 300MHz.

III. GROUP EXTENSIONS FOR MULTIPLE ECUS

In this section we start from a two-party protocol based on the Diffie-Hellman key exchange and further extend it to larger groups of ECUs. We also discuss possible variations of the protocol, highlighting advantages and disadvantages with regards to security and performance. Table II provides a summary for the notations used throughout our work which will be explained in the following paragraphs.

TABLE II
SUMMARY OF NOTATIONS

ECU	Electronic Control Unit
LECU	Logical ECU, obtained from merging two entities
SoECU	Security Orchestrator ECU
n	number of ECUs in the network
gsk	group secret key
msk	master secret key
$\{m\}_k$	symmetric encryption of m with key k
KDF	key derivation function
$\mathsf{Sig}_{\mathsf{ECU}}(m)$	digital signature performed by ECU on m

A. The two-party key exchange protocol based on STS

The key exchange protocols that are compatible with our implementation follow the generic Diffie-Hellman construction [44] which is arguably the most widely used key exchange protocol in computer networks. While its original embodiment [44] is vulnerable to man-in-the-middle attacks, subsequent variants such as the Station to Station (STS) protocol [10] remove this vulnerability by signing the exchanged key-shares. Since STS or other variations are implemented in many security suites (e.g. IPSec), we also chose it as basic primitive in the group key exchange protocol.

Figure 4 illustrates the steps of the STS protocol as we implement them for a key agreement between two ECUs. Compared to the original embodiment of the protocol, we require every key exchange to be authorized by a trusted authority within the system denoted as *Security Orchestrator* ECU, i.e., SoECU. Hence, the protocol start is not triggered by either of the participants but by SoECU which sends an orchestration message announcing the key exchange between two ECUs. The orchestration message contains a digital signature computed on the concatenation of the CAN Extended ID (CANxID) and a timestamp *ts* (we assume that nodes keep a common local time for the network which is also enforced by current AUTOSAR standards for time synchronization). This approach prevents impersonation or replay attacks directed towards SoECU. As a natural speedup, we also require that all computational steps are executed in parallel by the ECUs when triggered by SoECU to begin a key exchange. After receiving and verifying the signature announcing the key exchange

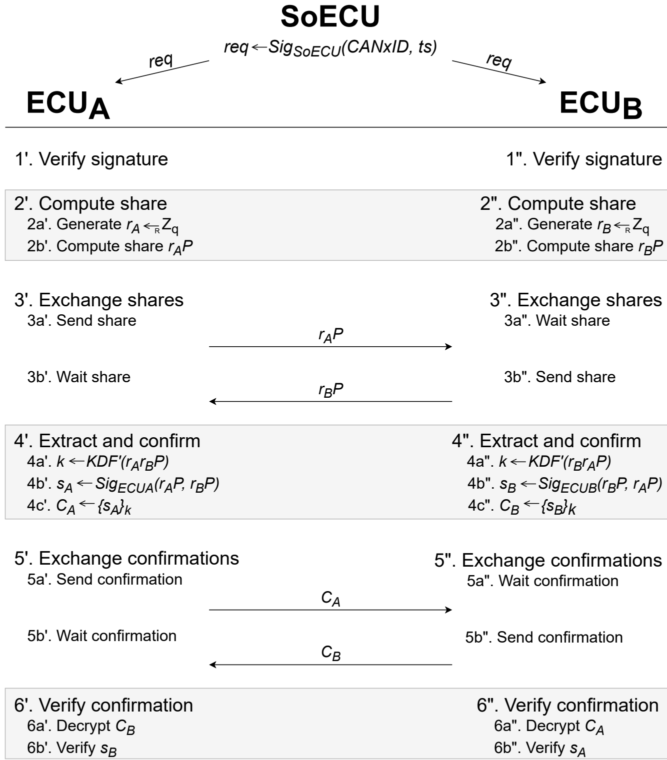


Fig. 4. Key exchange between $\mathsf{ECU_A}$ and $\mathsf{ECU_B}$ based on the STS protocol at the request of SoECU

requests (step 1), the ECUs will generate random integers, i.e., r_A and r_B, respectively (step 2a), that are further used to compute the Diffie-Hellman protocol shares, i.e., r_AP and r_BP (step 2b). The shares are exchanged between the two ECUs over CAN and neither of them will proceed to the next step until this transmission phase ends (steps 3a and 3b). Subsequently, both ECUs extract the shared session key as $k = \mathsf{KDF}'(r_Ar_BP)$ and $k = \mathsf{KDF}'(r_Br_AP)$ (step 4a), where $\mathsf{KDF}' : \{0, 1\}^* \rightarrow \{0, 1\}^\ell$ is a key derivation function and ℓ is the security level, e.g., 128 bits. Then the ECUs will digitally sign the shares as $s_A = \mathsf{Sig}_{\mathsf{ECU_A}}(r_AP, r_BP)$ and $s_B = \mathsf{Sig}_{\mathsf{ECU_B}}(r_BP, r_AP)$ (step 4b), and then encrypt these digital signatures in order to obtain the protocol confirmations, i.e., $C_A = \{s_A\}_k$ and $C_B = \{s_B\}_k$, respectively (step 4c). After exchanging the confirmations via CAN with the same mentions as before (steps 5a and 5b), both ECUs decrypt them and verify if the signatures are correct (steps 6a and 6b). Finally, the protocol is successful if and only if all signature verifications pass, otherwise the protocol is aborted.

One additional note is that since the amount of data sent on the bus must be carried by CAN-FD frames, i.e., with maximum 64 bytes in the data-field, our implementation requires four messages to be exchanged between the participants (compared to three in the original description of STS). Also, we omit sending the digital certificates explicitly during key exchanges in order to prevent unnecessary bus traffic each time the protocol is run. Due to the typically limited size of CAN networks and compact representations of elliptic curve points, we assume that each ECU stores local copies of all public keys (or certificates) belonging to the other units from its sub-network.

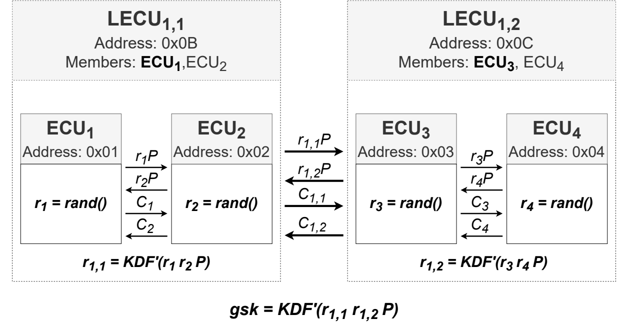


Fig. 5. Group key exchange between four units, formation of two LECUs

B. Formation of Logical ECUs

As stated, we require one trustworthy ECU, i.e., the *security orchestrator* SoECU, to orchestrate and monitor the key exchange between the ECUs. Furthermore, we also assign other operations to the security orchestrator, e.g., revoking the credentials of a node in case it becomes corrupted or merging two networks. A convenient practical instantiation for the SoECU may be a gateway ECU. The initial configuration of the CAN network is a fixed set of n units connected to the bus, referred as physical units, or simply ECUs, and denoted as $\mathsf{ECU}_i, i = 1..n$. To avoid conflicts between ECUs during the key negotiation, we assign a fixed, unique *address* encoded as a one-byte integer to each ECU. Adding addresses to ECUs on the bus is already present in upper layer specifications of the CAN bus such as J1939-21 [45] and does not interfere with existing specifications. The identifier fields of the orchestration messages and the responses provided by protocol participants will specify the role of the message as well as the sender and receivers. Figure 5 shows an example of a CAN network with four such physical ECUs ($n = 4$), denoted as ECU_1, ECU_2, ECU_3 and ECU_4 having the addresses *0x01*, *0x02*, *0x03* and *0x04*, respectively.

We build the group key exchanges on the CAN bus over the primitive operation of *merging two ECUs* by engaging them in a key exchange. In the example from Figure 5, ECU_1 is merged with ECU_2, while ECU_3 is merged with ECU_4. These operations are requested by the SoECU using the ECU physical addresses that were previously introduced, and can be performed in parallel. During the execution, each physical unit $\mathsf{ECU}_i, i = 1..n$ generates an ephemeral random value r_i for computing the Diffie-Hellman protocol share as r_iP. At the end of these exchanges, two subgroups are formed. We refer to these as *logical ECUs* and denote them as $\mathsf{LECU}_{l,j}$, where l indicates the level of the controller and j is the index of the unit within level l. Consequently, a LECU is an abstract ECU which allows a subgroup of physical ECUs to be represented and addressed as a single entity. It is characterized by: i) the physical ECUs which are members of the LECU ii) a unique, logical address encoded as a one-byte integer for addressing the subgroup and iii) a value $r_{l,j}$ (initially

set to null) representing the most recently established secret between the members. In the example from Figure 5 both LECUs, i.e., $\mathsf{LECU}_{1,1}$ and $\mathsf{LECU}_{1,2}$, have two members and the secret values shared within the subgroups are $r_{1,1}$ and $r_{1,2}$, respectively. Subsequently, we extend the merging operation to also support LECUs as operands.

Whenever two LECUs need to be merged, the security orchestrator uses their addresses in order to initialize the operation. All member ECUs from both sides will be triggered by this event, but only one physical device from each subgroup will act and conduct the key exchange protocol on behalf of the LECU. We call this unit the *active member* of the LECU, while the other units are *passive members*. For practical purposes, the active member of a logical unit can be designated as the ECU with the highest computational resources. Since the passive members must also extract the newly agreed secret value, we require that the active member of $\mathsf{LECU}_{l,j}$ will always use the corresponding subgroup secret, i.e., $r_{l,j}$, as the private key for computing the Diffie-Hellman protocol share. Thus, the passive members can monitor the key exchange messages through the CAN bus, while also extracting the agreed secret shared key. The result of merging two LECUs will then be a new logical unit which contains all the members from both operands and a secret key established between them. In the example from Figure 5, $\mathsf{LECU}_{1,1}$ and $\mathsf{LECU}_{1,2}$ are merged at SoECUs request using the addresses *0x0B* and *0x0C*. One member from each LECU, i.e., ECU_1 in $\mathsf{LECU}_{1,1}$ and ECU_3 in $\mathsf{LECU}_{1,2}$, is designated as the active member (highlighted in bold) and both protocol shares are computed using the LECU secret values as $r_{1,1}P$ and $r_{1,2}P$, respectively. Finally, the result of this merging operation is a LECU which contains all ECUs from the network as members having the group secret key $gsk = \mathsf{KDF}'(r_{1,1}r_{1,2}P)$, where $\mathsf{KDF}' : \{0,1\}^* \to \{0,1\}^\ell$ is a key derivation function with security level ℓ.

C. The main group key exchange scheme and variations

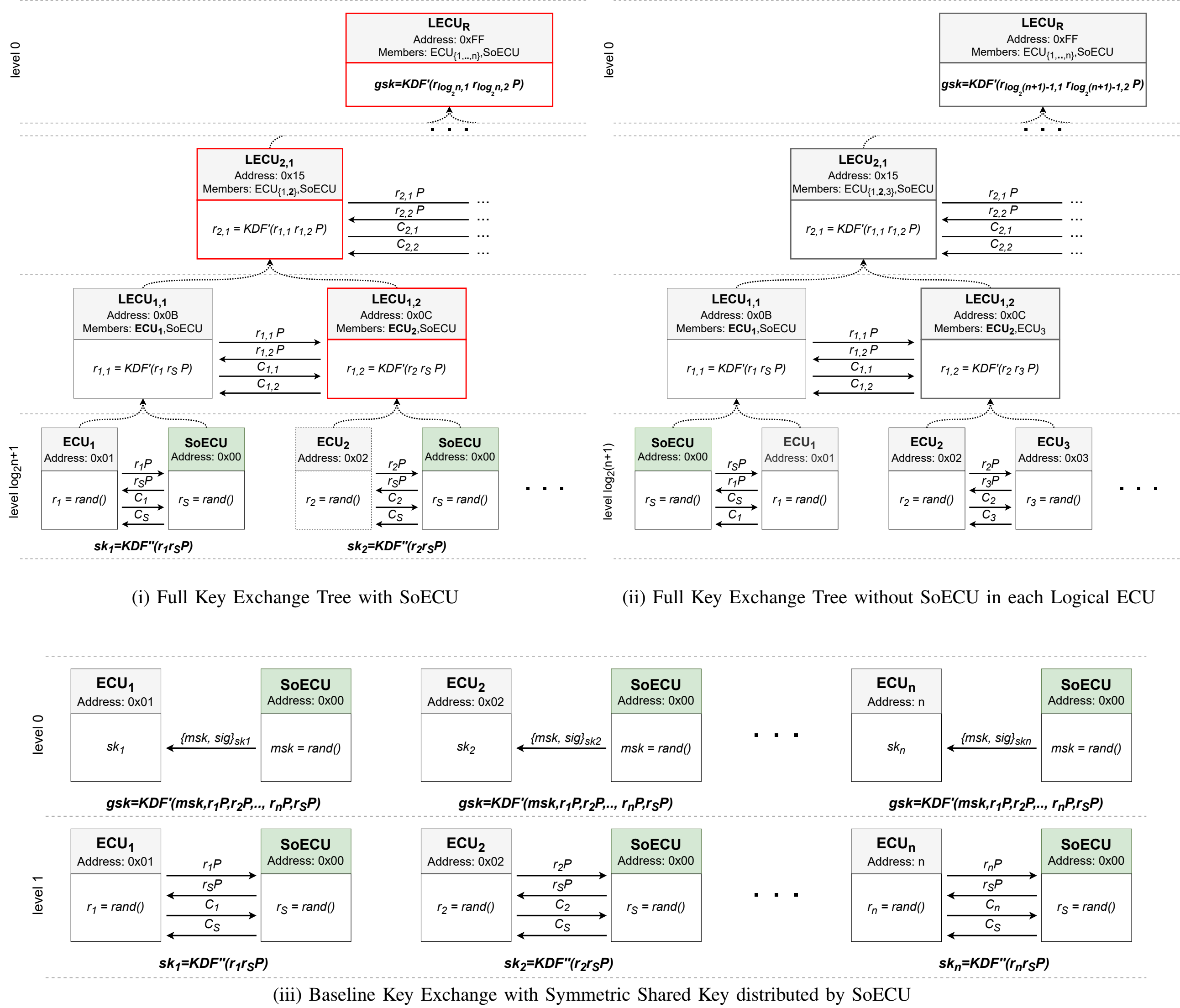
We now detail the main group key exchange scheme which uses the previous rationale of merging two physical or logical entities. Due to obvious concerns related to computational and communication resources, we further simplify the main scheme and introduce two variations of it that are more efficient. The three resulting group key exchange schemes are summarized in what follows.

1) Full Key Exchange Tree with SoECU*:* is the main version of the scheme with one key exchange between each ECU and SoECU, i.e., building one logical ECU between SoECU and each other ECU, followed by pair-wise key exchanges between logical ECUs up to the root of the tree. In this way the group key exchange problem is solved recursively by decomposing it into smaller problems. The inclusion of the SoECU in a subgroup with each ECU is relevant for at least three reasons: i) in this way the security orchestrator shares a secret key with each ECU that can be used for establishing a private communication channel, ii) all the subsequent key agreements between logical ECUs can be monitored by the security orchestrator and iii) the security orchestrator contributes with fresh random material to each key, removing potential weaknesses in case that some controller unit has poor random number generators.

Figure 6 (i) illustrates the main protocol scheme in the context of a group key exchange for a network with n regular ECUs denoted as $\mathsf{ECU}_i, i = 1..n$, plus a trustworthy unit assigned with the role of the SoECU. The protocol follows the structure of a binary tree, where the root is the logical controller LECU_R containing all the network units as members. In this representation, each level of the tree is a protocol step in which merging operations are conducted by physical ECUs in parallel, so the overall runtime is in fact logarithmic in the number of ECUs. The only exception is the bottom level of the tree, which includes SoECU in each key exchange (thus it is non-parallelizable). Here, each ECU_i is merged with SoECU in order to form a logical controller and extract two secrets. The first one $r_{l,j} = \mathsf{KDF}'(r_i r_s P)$ is the secret value assigned to the logical controller $\mathsf{LECU}_{l,j}$ in the next level of the tree and the second one $sk_i = \mathsf{KDF}''(r_i r_s P)$ is a secret key used for establishing a private communication channel between ECU_i and SoECU. Here $\mathsf{KDF}' : \{0,1\}^* \to \{0,1\}^\ell$ and $\mathsf{KDF}'' : \{0,1\}^* \to \{0,1\}^\ell$ are two distinct key derivation functions with security level ℓ. From the third level of the tree, merging operations are performed between logical controllers which include the security orchestrator SoECU as a member (the SoECU is an active member only when there is no other physical ECU in the LECU).

2) Full Key Exchange Tree without SoECU *in each Logical ECU:* is a variation in which we cut the last level of the tree and let the ECUs exchange keys independently, i.e., the ECUs are clustering in the last level under logical ECUs without SoECU being present in each logical ECU. SoECU is still responsible for orchestrating the key exchange. Since the last level of the tree is the largest, this should significantly reduce the computational and communication overheads. One disadvantage is that SoECU will no longer have a shared secret key with each ECU, but SoECU will group itself as a regular node in the key exchange (for this reason we now have $n+1$ nodes in the key exchange) and will be in possession of the group secret key. This version is illustrated in Figure 6 (ii) and since it is similar to the previous, further details shall be superfluous.

3) Baseline Key Exchange with Symmetric Shared Key distributed by SoECU*:* is the version in which we try to remove some of the expensive elliptic curve operations. In this version we consider that SoECU performs a key exchange with each of the ECUs, i.e., the last level of the main scheme tree is fully formed, and subsequently sends the master secret key msk from which the group key is derived. This value will be signed and independently encrypted by SoECU with the secret key shared with each of the ECUs. In this case however, logical ECUs are formed only between SoECU and each other ECU, i.e., logical ECUs from the upper layers are missing. This results in a more rigid key distribution scheme that will not allow subgroups of ECUs to communicate under a distinct secret key, i.e., under a logical ECU. Once an ECU is removed, a new session key must be shared by the same symmetric key exchange procedure. This version



(i) Full Key Exchange Tree with SoECU

(ii) Full Key Exchange Tree without SoECU in each Logical ECU

(iii) Baseline Key Exchange with Symmetric Shared Key distributed by SoECU

Fig. 6. Group key exchange schemes

of the scheme is illustrated in Figure 6 (iii). To ensure that each ECU has contributed to the group secret key, this key is derived from all the key material that has been exchanged on the bus, i.e., $\mathsf{gsk} = \mathsf{KDF}'(msk, r_1P, r_2P, ..., r_nP, r_SP)$. Here msk is the master secret key generated by SoECU. This random value is encrypted along with a signature on it, i.e., $sig = \mathsf{Sig}_{\mathsf{SoECU}}(msk, r_1P, r_2P, ..., r_nP, r_SP)$. In this way, all ECUs contribute to the group secret key with their random material and SoECU confirms to each ECU that the common group key is built with its random secret material msk.

D. Removing nodes from the network

In the rare event of an ECU becoming corrupted, since physical removal of the node in a timely manner is not always possible, we also need to provide procedures for logical removal of the node from the network. Whenever an ECU is removed from the network, the group secret key must be renewed such that the removed node will be unable to retrieve it. In our implementation, the removed ECU will be replaced by the security orchestrator SoECU in order to maintain the tree structure and avoid minimizing the entropy of the group secret key. Consequently, ECU removal is easily orchestrated by the SoECU, by re-running all the key exchanges from the last LECU that includes the corresponding physical ECU on the path that leads to the root of the tree. An example for such path is highlighted in red in Figure 6 (i) for the main scheme. Here, ECU_2 is removed by renewing the secret values from $\mathsf{LECU}_{1,2}$, $\mathsf{LECU}_{2,1}$ up to the root LECU_R. If a removed ECU has to be added back to the network, then the security orchestrator SoECU will re-include the node in the logical LECU by triggering a new key exchange for the LECUs. The key exchange will propagate further to the root of the tree along the same path that led to the removal. For brevity, further details on this procedure are deferred to Appendix A.

IV. Implementation details

In this section we present our experimental setup and provide the implementation details.

TABLE III
HARDWARE CHARACTERISTICS OF THE EMPLOYED PLATFORMS

Characteristic	TC224	TC277	TC297	SAM V71
Cores	1	3	3	1
Architecture	32-bit	32-bit	32-bit	32-bit
Max. frequency	133 MHz	200 MHz	300 MHz	300 MHz
Flash	1 MB	4 MB	8MB	2 MB
RAM	96 KB	472 KB	728 KB	2 MB
EEPROM	96 KB	384 KB	768 KB	256 KB
CAN-FD	Yes*	Yes*	Yes*	Yes

*CAN-FD supported by microcontroller, but not by the on-board transceivers of the evaluation kits

A. Automotive-grade hardware support

We evaluate the performance of the proposed mechanism using a series of 32-bit automotive-graded platforms. These include three Infineon evaluation boards based on the TC224, TC277 and TC297 microcontrollers, as well as two Microchip SAM V71 Xplained Ultra evaluation boards based on the ARM Cortex M7. Table III highlights the key characteristics of each device.

The Infineon microcontrollers from our experiments are members of the AURIX family of devices, designed for high performance and reliable automotive applications. While the TC224 features a single core that maxes out at a frequency of 133MHz, the other two family members offer three-core architectures that can reach clock frequencies of up to 200MHz in case of the TC277 and 300MHz in case of the TC297. The ARM-based SAM V71 features a single-core processor that also reaches a top frequency of 300MHz. Specialized cryptographic peripherals, e.g., true random number generator and hardware accelerated AES engines are provided by both microcontroller families. While these peripherals are easily accessible on the SAM V71 boards, the Infineon platforms incorporate them in dedicated Hardware Security Modules (HSM) and the use of these modules is reserved for industry applications. As a consequence, when implementing the protocol on Infineon's platforms, we opted towards software-based implementations of the required cryptographic primitives (i.e. pseudo-random generator, AES encryption/decryption, etc.). Fortunately, these symmetric operation come with very small performance penalties compared to the public-key operations required by the key exchange.

Regarding connectivity, all platforms are CAN-FD compatible. The SAM V71 devices are equipped with on-board ATA6561 CAN-FD ready transceivers and thus can be directly connected using the dedicated CAN-H and CAN-L pins and a twisted pair of wires as shown in Figure 7 (i). Since both nodes have the same computational power, the network defined by them is further used by us as a homogeneous network example. In contrast, all Infineon boards have different computational capabilities and include only high speed CAN transceivers which do not support higher bit rates specific to CAN-FD. In order to enable CAN-FD communication using the AURIX microcontrollers, we connected external CAN-FD transceivers to the TC277 and the TC297 boards, while the third one, i.e., the TC224, was used only for individual computational benchmarks. The transceivers that we used are NCV7344 from ON Semiconductor, available as off-the-shelf solution from Mikroelektronika on CAN FD Click boards which are equipped with interface pins, 120Ω termination resistors and differential bus connectors. Figure 7 (ii) illustrates the circuit diagram for connecting the AURIX-based microcontrollers (left) as well as the resulting setup (right) that was used as an example of a network with heterogeneous nodes.

During the performance analysis stage, each microcontroller was configured to operate at its maximum rated clock speed. Further, for measuring the execution time, we used a logic analyzer from Saleae as shown in both Figures 7 (i) and 7 (ii) in conjunction with high-speed toggling pins from each board. For increased precision, the analyzer was configured to operate at the sampling rate of 50MS/s.

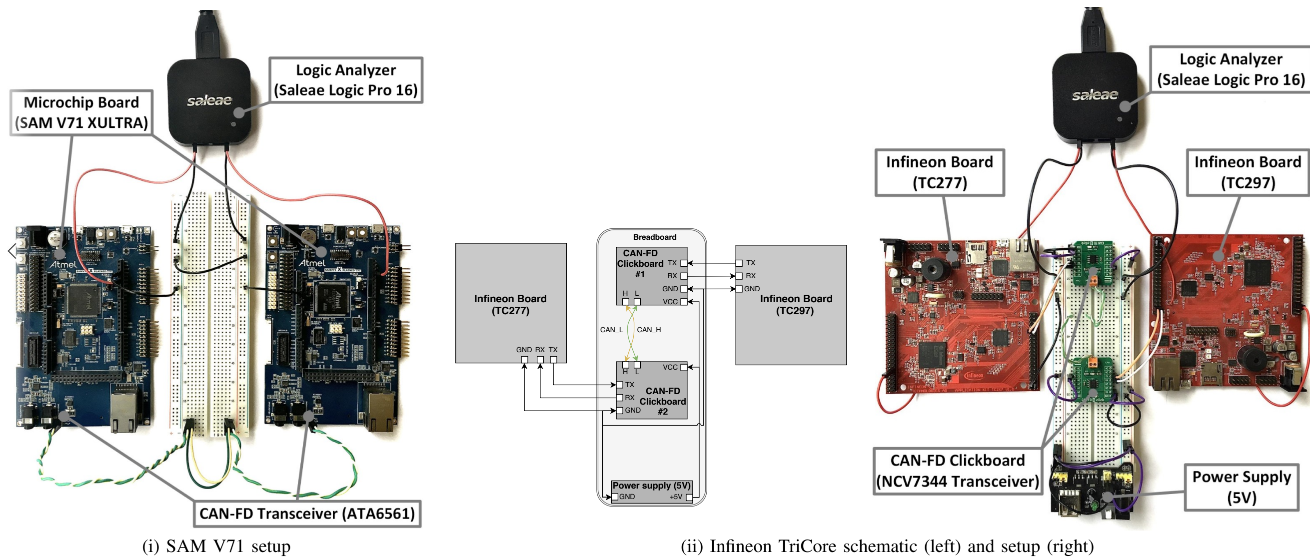
B. AUTOSAR-compliant software implementation

Our implementation follows existing AUTOSAR recommendations for cryptographic services. In Figure 8 we show the architecture of our implementation in the light of the AUTOSAR specifications. Here, the Group Key Manager (GKM) is the application software component that manages the activity of the ECU during group key management operations. FourQlib and other hardware-based implementations of cryptographic primitives are part of the Crypto Driver Objects. The GKM can request cryptographic operations from the Crypto Driver Objects through the standardized AUTOSAR software cryptographic stack which includes: Crypto Service Manager, Crypto Interface and Crypto Driver (https://www.autosar.org/).

The implementation process, abstracted in Figure 9, started with the migration of the required cryptographic libraries on top of the Hardware Abstraction Layer (HAL). Firstly, Microsoft's Four$\mathbb{Q}$ library (https://github.com/microsoft/FourQlib) was ported on our controllers in order to benchmark the protocol performance when implemented based on the faster Four$\mathbb{Q}$ elliptic curve. Afterwards, in order to obtain relevant comparison data, we also ported RELIC's cryptographic toolkit (https://github.com/relic-toolkit/relic) with the purpose of evaluating the protocol when implemented based on a similarly sized elliptic curve, namely NIST's P-256. Both of these libraries offer native support for 32-bit architectures and, whenever possible, were configured to rely on the available cryptographic hardware support. Furthermore, both libraries provide dedicated API functions for performing the ECDH protocol steps as well as for elliptic curve digital signature schemes (Schnorr's digital signature in case of Microsoft's Four$\mathbb{Q}$ library and ECDSA in case of RELIC's toolkit). These API functions were then uniformly wrapped at the *Station to Station (STS) Protocol* layer. Finally, the GKM software component was implemented at the *Application* layer as a state machine that follows the execution flowchart shown in Figure 13, which will be further used for measuring the protocol runtime.

C. Protocol message orchestration

The SoECU is responsible for orchestrating group key exchanges, i.e., coordinating a sequence of two-party key exchanges until all ECUs agree on a group secret key. This is



(i) SAM V71 setup

(ii) Infineon TriCore schematic (left) and setup (right)

Fig. 7. Experimental setup with: two homogeneous ECUs, SAM V71 boards connected via on-board ATA6561 CAN transceivers (i) and two heterogeneous ECUs, Infineon TC277/TC297 boards connected via external CAN-FD transceivers (ii)

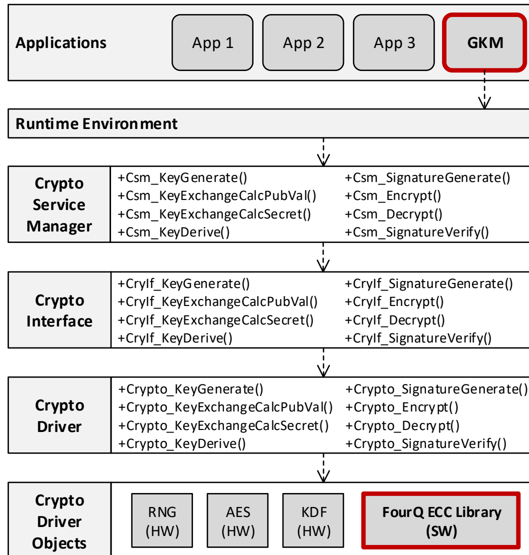


Fig. 8. Overview of our group key exchange implementation from the AUTOSAR requirements

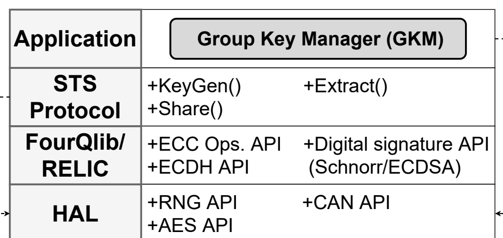


Fig. 9. Application stack: hardware, cryptographic libraries and the Group Key Manager

achieved in our implementation by allocating dedicated 29-bit CAN message identifiers which incorporate ECU addresses as well as other protocol metadata. The integration of specific node addresses in 29-bit CAN identifiers is also specified by

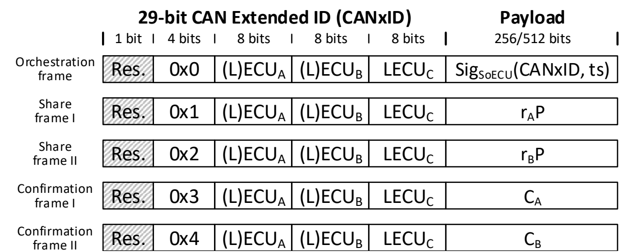


Fig. 10. Structure of protocol frames

J1939-21 standard for trucks and commercial vehicles [45].

The protocol frames are designed to include both orchestration data and protocol payloads, so that the busload is kept as low as possible. A detailed view is shown in Figure 10. We use four bit-fields for defining the 29 bit extended CAN IDs (CANxID). The first field, following the reserved bit, requires 4 bits to encode the key exchange communication steps as shown in Figure 4. The three remaining fields encode the addresses of the participants and of the resulting LECU. Note that the participants can be either individual units or LECUs. Consequently, there are five frames required by each individual key exchange, i.e., the key exchange orchestration frame sent by the SoECU, the two frames containing the key shares and the two confirmation frames sent by the participants.

V. Experimental Results

In this section we provide comparative results related to both computational and bus requirements for the two-party key exchange scenario. We then use these results to estimate the performance of the group key exchange protocol for an arbitrary number of nodes.

A. *Performance evaluation for cryptographic primitives*

As a first step in our performance benchmark, we measured the execution time of each elliptic curve library function

TABLE IV
ECDH AND SCHNORR'S DIGITAL SIGNATURE BENCHMARK RESULTS WITH FOUR$\mathbb{Q}$ ELLIPTIC CURVE

	Operation	Measured execution time (ms)			
		TC224	TC277	TC297	SAM V71
ECDH	Share	4.99	4.03	1.92	17.01
	Extract	12.53	8.39	4.79	54.32
Schnorr DS	Gen	5.11	4.12	1.98	17.06
	Sign	6.12	4.85	2.36	18.96
	Verify	17.54	12.45	6.73	70.99

TABLE V
ECDH AND ECDSA BENCHMARK RESULTS WITH P-256 ELLIPTIC CURVE USING RELIC LIBRARY IMPLEMENTATION

	Operation	Measured execution time (ms)			
		TC224	TC277	TC297	SAM V71
ECDH	Share	25.43	21.46	12.83	11.5
	Extract	60.67	50.75	30.03	27.67
ECDSA	Gen	25.43	23.61	14.82	11.33
	Sign	29.54	25.9	14.31	13.01
	Verify	80.01	54.11	30.85	35.53

that was used during the protocol implementation. These measurements were performed on all of the platforms that were previously described in section IV-A and the numerical results are shown in Tables IV and V. The former table contains measurements that were obtained when evaluating the primitives based on the Four$\mathbb{Q}$ elliptic curve, while the latter shows the results that were obtained when using NIST's P-256 elliptic curve implemented in RELIC. In both tables, the first column indicates the cryptographic primitives being evaluated and the second one contains the corresponding operations for each of them. The remaining four columns present the measured execution time, in milliseconds, that was obtained for each tested device. Further, the results from both tables were grouped by cryptographic primitives, i.e., ECDH and digital signature (DS), and a visual representation of them is provided in Figures 11 and 12.

Analyzing our results, we find that in the context of the Infineon devices all functions from both cryptographic primitives are executed significantly faster when using Four$\mathbb{Q}$ as the underlying elliptic curve. A rough estimation places the more affordable TC224-based device in the area of five times faster execution speed when implementing Four$\mathbb{Q}$, while the higher end TC277 and TC297-based devices benefit from speed improvements that surpass five and six times, respectively. Interestingly, in the context of the ARM-based microcontroller the result differs, i.e., it is approximately two times faster when implementing the cryptographic primitives based on NIST's P-256 elliptic curve. The most likely explanation of this observed behavior is the active maintenance of RELIC's software implementation which offers greater compatibility with the optimization engine provided by the ARM compiler. Regardless, Four$\mathbb{Q}$ on Infineon TC297 is the top performer and still at least five times faster than P-256 on ARM.

B. Protocol performance: the two-party case

The next step that we take towards evaluating the group key exchange protocol performance is to evaluate the protocol

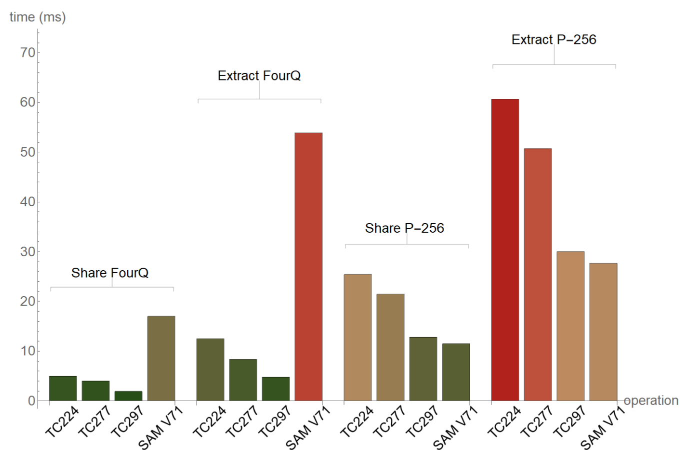


Fig. 11. Computational time for the Diffie-Hellman key exchange with Four$\mathbb{Q}$ vs. NIST P-256 on the four development boards

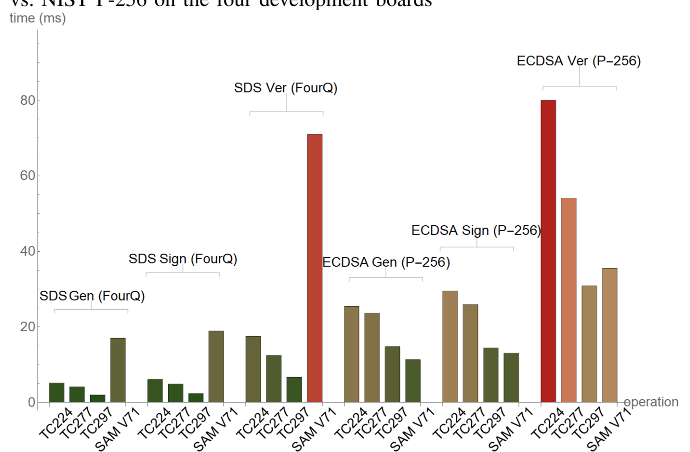


Fig. 12. Computational time for SDS (Four$\mathbb{Q}$) vs. ECDSA (NIST P-256) on the four development boards

runtime between two ECUs.

Figure 13 shows the flowchart associated to the protocol implementation that is followed by any two ECUs when engaged in a key exchange. In this representation, each individual protocol step is characterized by the execution time that it induces, allowing us to easily separate and group two types of overheads: the computational overhead $\mathbf{T}_{\mathsf{ECU}}$ and the communication overhead $\mathbf{T}_{\mathsf{BUS}}$. By grouping the individual overhead components along the three stages depicted in Figure 13, the duration of a complete key exchange (denoted as $\mathbf{T}_{\mathbf{kex}}$) can be expressed as the sum of the computational and bus time from the first two protocol stages plus the computational time required to decrypt and verify the signature in the third stage of the protocol, i.e.,

$$\mathbf{T}_{\mathbf{kex}} = \underbrace{\mathbf{T}^{(1)}_{\mathsf{ECU}} + \mathbf{T}^{(1)}_{\mathsf{BUS}}}_{\text{1st stage}} + \underbrace{\mathbf{T}^{(2)}_{\mathsf{ECU}} + \mathbf{T}^{(2)}_{\mathsf{BUS}}}_{\text{2nd stage}} + \underbrace{\mathbf{T}^{(3)}_{\mathsf{ECU}}}_{\text{3rd stage}}$$

We further use the notation $\mathbf{T}_{\blacklozenge}$, where $\blacklozenge \in \{ver, shr, key, sig, enc, dec\}$ indicates the computational overhead for verifying a digital signature, computing the Diffie-Hellman

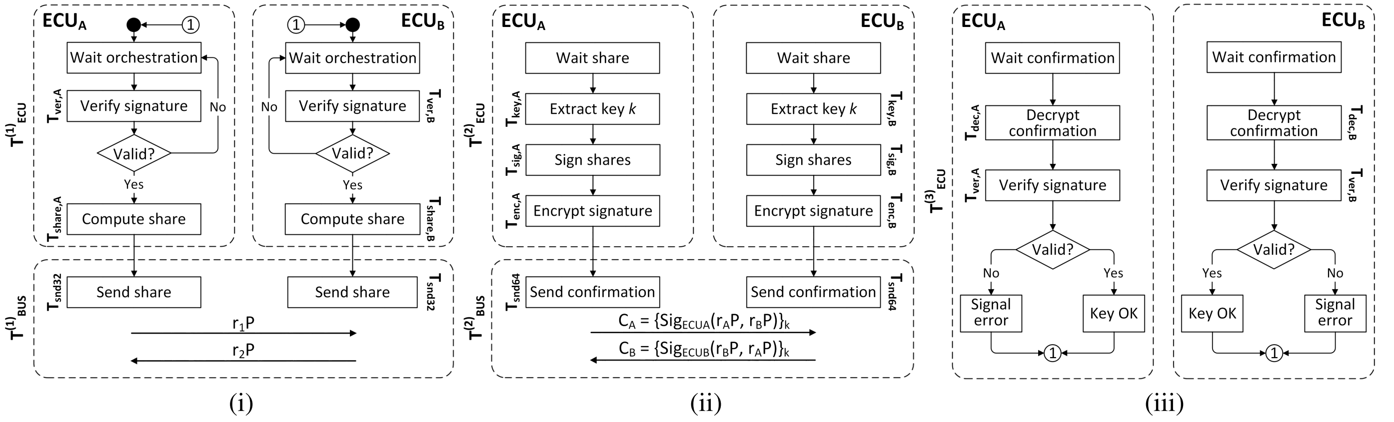


Fig. 13. Flowchart of a key exchange between two ECUs as implemented in our experimental setup and divided into three stages: computing and exchanging the Diffie-Hellman shares (i), extracting the secret key and exchanging the confirmations (ii) and validating confirmations (iii)

share, extracting the secret key, computing a digital signature, encrypting and decrypting the signature, respectively. This allows us to expand the three computational overheads as:

$$\mathbf{T}^{(1)}_{\mathsf{ECU}} = \mathbf{T}_{ver} + \mathbf{T}_{shr} \qquad \mathbf{T}^{(2)}_{\mathsf{ECU}} = \mathbf{T}_{key} + \mathbf{T}_{sig} + \mathbf{T}_{enc}$$

$$\mathbf{T}^{(3)}_{\mathsf{ECU}} = \mathbf{T}_{dec} + \mathbf{T}_{ver}$$

This expansion assumes that the two ECUs are performing the computations in parallel and have the same computational power/availability. However, we also need to account for the possibility of having a heterogeneous network, i.e., nodes with distinct computational power. In this case, the time estimation must always account for the slowest device from the setup and use its timing to further estimate the protocol runtime. That is, the computational terms from the previous equation should be computed as the maximum runtime values resulting from the two protocol participants: $\mathbf{T}_{\blacklozenge} = \max(\mathbf{T}_{\blacklozenge,A}, \mathbf{T}_{\blacklozenge,B})$.

Further, the delays induced by the CAN bus are denoted as $\mathbf{T}_{\mathsf{snd32}}$ and $\mathbf{T}_{\mathsf{snd64}}$, indicating the amount of time required to send 32 and 64-byte data frames. Since the CAN bus allows only one frame to be sent at a time (with up to 64 bytes data-field for CAN-FD), the communication overhead can be expressed as the time for two frame transmissions of 32 and 64 bytes, respectively:

$$\mathbf{T}^{(1)}_{\mathsf{BUS}} = 2 \times \mathbf{T}_{\mathsf{snd32}} \qquad \mathbf{T}^{(2)}_{\mathsf{BUS}} = 2 \times \mathbf{T}_{\mathsf{snd64}}$$

Note that the first protocol stage requires only the Diffie-Hellman key shares to be sent which are two times smaller than the digital signature from the second stage.

As concrete examples and proofs of correctness for the previously described estimation methodology, in Tables VI and VII we show the measured execution time of Four$\mathbb{Q}$-based key exchanges between two nodes connected via CAN-FD. In the first example the network is homogeneous with the nodes being instantiated by two identical SAM V71 boards, while the second example shows a heterogeneous network composed of two distinct platforms, i.e., the TC297 and TC277 respectively. In both tables, the first column indicates the protocol step according to Figure 4. Subsequently, the second column indicates

TABLE VI
STS w/ Four$\mathbb{Q}$ protocol runtime on CAN-FD, measured in a homogeneous network with two SAM V71 boards

Protocol step	Notation	Measured execution time	
		SAM V71-A	SAM V71-B
1. Verify signature	T_{ver}	$70.55ms$	$70.55ms$
2. Compute share	T_{shr}	$16.74ms$	$16.74ms$
3a. Send share (I)	T_{snd32}	$258us$	
3b. Send share (II)	T_{snd32}	$258us$	
4a. Extract secret key k	T_{key}	$54.19ms$	$54.2ms$
4b. Sign shares	T_{sig}	$18.98ms$	$18.98ms$
4c. Encrypt signature	T_{enc}	$28us$	$21us$
5a. Send confirmation (I)	T_{snd64}	$396us$	
5b. Send confirmation (II)	T_{snd64}	$396us$	
6a. Decrypt confirmation	T_{dec}	$21us$	$21us$
6b. Verify signature	T_{ver}	$70.95ms$	$70.95ms$
Total time (measured)	T_{kex}	$232.8ms$	$232.7ms$

the associated execution time in the previously introduced notation and the last two columns contain the values obtained by experimental measurements. The timings are consistent with already provided measurements from Table IV. To conclude with, in the left plot from Figure 14 we graphically depict the measured computational time for heterogeneous and homogeneous networks. Then, in the right plot, we contrast it with the estimated time for a homogeneous network based on any of the four controllers from our experiments. The protocol runtime between the TC297 and TC277 is $43.94ms$ for the Four$\mathbb{Q}$ version which is almost six time faster than the P-256 curve in case of the SAM boards which required $232.79ms$. This allows us to infer that the runtime between two identical TC297 controllers will be around $24ms$.

C. Performance of the group key exchange

We now consider to synthetically evaluate the performance of the group key exchange when an arbitrary number of ECUs is present on the bus. Specifically, for the case of n controllers, the efficiency of the three envisioned protocols can be synthetically expressed as follows:

1) Full Key Exchange Tree with SoECU, i.e., the main version of the scheme, has a total runtime upper bounded by:

TABLE VII
STS W/ FOUR$\mathbb{Q}$ PROTOCOL RUNTIME ON CAN-FD, MEASURED IN A HETEROGENEOUS NETWORK WITH TWO INFINEON BOARDS

Protocol step	Notation	Measured execution time TC297	TC277
1. Verify signature	T_{ver}	$6.75ms$	$12.49ms$
2. Compute share	T_{shr}	$1.92ms$	$4.06ms$
3a. Send share (I)	T_{snd32}	$210us$	
3b. Send share (II)	T_{snd32}	$210us$	
4a. Extract secret key k	T_{key}	$4.79ms$	$8.38ms$
4b. Sign shares	T_{sig}	$2.4ms$	$4.95ms$
4c. Encrypt signature	T_{enc}	$86us$	$179us$
5a. Send confirmation (I)	T_{snd64}	$340us$	
5b. Send confirmation (II)	T_{snd64}	$340us$	
6a. Decrypt confirmation	T_{dec}	$121us$	$240us$
6b. Verify signature	T_{ver}	$6.77ms$	$12.54ms$
Total time (measured)	T_{kex}	$38.31ms$	$43.99ms$

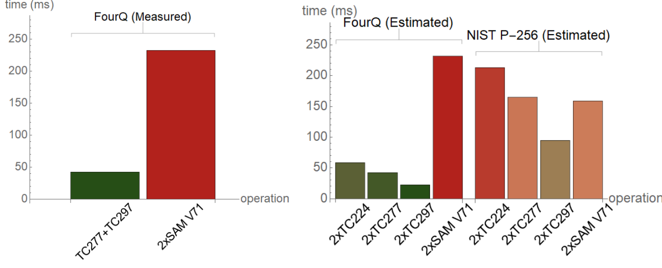


Fig. 14. Protocol runtime between 2 nodes: measured heterogeneous/homogeneous network (left) vs. synthetic estimation for homogeneous networks (right)

$$\mathbf{T}_{\mathbf{gkex}}^{\mathbf{main}} = (n + \log_2 n)\mathbf{T}_{\mathsf{ECU}}^{1:3} + (2n-1)\mathbf{T}_{\mathsf{BUS}}^{1:2}$$

Without loss of generality we assume that n is a power of two. If this is not the case, the base 2 logarithm will get ceiled to the nearest integer. This relation accounts for computations that are done in parallel by the nodes but the bus transmission cannot be parallelized. That is, we assume that $n\mathbf{T}_{\mathsf{ECU}}^{1:3}$ computations are required in the last level of the binary tree, i.e., between the SoECU and each other ECU. From the second last level upwards, operations can be done in parallel by the logical ECUs which requires only $(\log_2 n)\mathbf{T}_{\mathsf{ECU}}^{1:3}$ computations. The bus however is a common resource and nodes cannot send messages at the same time. Consequently, there is one key exchange message set to be sent on the bus for each pair of nodes in the binary tree, i.e., $2^{\log_2 n+1} - 1 = 2n - 1$.

2) Full Key Exchange Tree without SoECU *in each Logical ECU*, i.e., the version in which we cut the last level of the tree and let the ECUs exchange keys independently from the SoECU, has a total runtime upper bounded by the relation below which basically omits the last level of the tree and adds 1 to the number of ECUs:

$$\mathbf{T}_{\mathbf{gkex}}^{\mathbf{w/o}} = \log_2(n+1)\mathbf{T}_{\mathsf{ECU}}^{1:3} + n\mathbf{T}_{\mathsf{BUS}}^{1:2}$$

3) Baseline Key Exchange with Symmetric Shared Key by SoECU *in all LECUs*, i.e., the variation of the scheme where a common group secret key is assigned by SoECU, has a total runtime upper bounded by the relation below:

$$\mathbf{T}_{\mathbf{gkex}}^{\mathbf{sym}} = n(\mathbf{T}_{\mathsf{ECU}}^{1:3} + \mathbf{T}_{\mathsf{BUS}}^{1:2}) + \mathbf{T}_{sig} + n\mathbf{T}_{enc} + n\frac{\mathbf{T}_{\mathsf{BUS}}^{1:2}}{2} + \mathbf{T}_{\mathsf{ECU}}^{(3)}$$

Here we assume the usual handshake with all n ECUs that is done sequentially by the SoECU, i.e., requiring $n(\mathbf{T}_{\mathsf{ECU}}^{1:3} + \mathbf{T}_{\mathsf{BUS}}^{1:2})$. To this, we add the time to compute the signature, i.e., $\mathbf{T}_{sig}$, and the time required to encrypt it for each other ECU, i.e., $n\mathbf{T}_{enc}$. Sending the encrypted signature and master secret key will require $n\frac{\mathbf{T}_{\mathsf{BUS}}^{1:2}}{2}$ since the transmission is only from the SoECU to the rest of the ECUs. Finally, the ECUs will need to decrypt it and verify the signature which requires $\mathbf{T}_{\mathsf{ECU}}^{(3)} = \mathbf{T}_{dec} + \mathbf{T}_{ver}$.

To avoid overloading the above relations, we did not include the overhead induced by the SoECU to sign and send the orchestration messages which is separately provided below:

$$\mathbf{T}_{\mathsf{SoECU}}^{\mathbf{main}} = (2n-1)(\mathbf{T}_{sig} + \mathbf{T}_{\mathsf{snd64}})$$

$$\mathbf{T}_{\mathsf{SoECU}}^{\mathbf{w/o}} = \mathbf{T}_{\mathsf{SoECU}}^{\mathbf{sym}} = n(\mathbf{T}_{sig} + \mathbf{T}_{\mathsf{snd64}})$$

The first relation reflects the case of the main version of the protocol where the full key exchange tree requires $2n-1$ handshakes. The second relation is for the last two protocol versions in which the SoECU participates either as a regular ECU or polls the nodes in a linear manner requiring only n orchestration messages, i.e., in the last two protocol versions the computations of the SoECU are halved. These computations can be further alleviated if the orchestration messages are computed in advance and buffered. Moreover, if the key exchange tree remains fixed and dynamic node addition/removal is not needed, SoECU can sign and send a single orchestration initialization message after which the nodes will proceed to exchange keys in a predefined manner.

Figure 15 provides a graphical depiction for the estimated runtime of the group key exchange protocol and its versions in case of a homogeneous network with $n = 2..32$ Infineon TC297 ECUs. First, we show the performance of the main protocol depending on the type of curve (i), i.e., Four$\mathbb{Q}$ vs. NIST P-256. The Four$\mathbb{Q}$ curve obviously leads to results that are roughly 5 times faster. Then we compare the three protocol versions (ii) and it seems that the third version of the scheme, i.e., the baseline version which replaces the upper layers of the tree with a symmetric key exchange, doesn't save so much of the overall protocol runtime. The second version however is much faster since it allows key exchanges to run in parallel on each level of the tree. Namely, in case of 32 ECUs, the main group key exchange scheme allows a common key to be computed in around $1.1s$ out of which $\mathbf{T}_{\mathsf{SoECU}}^{\mathbf{main}} \approx 0.17s$ and $\mathbf{T}_{\mathbf{gkex}}^{\mathbf{main}} \approx 0.9s$. The second protocol version reduces this to $\mathbf{T}_{\mathsf{SoECU}}^{\mathbf{w/o}} \approx 0.1s$ and $\mathbf{T}_{\mathbf{gkex}}^{\mathbf{w/o}} \approx 0.17s$. While in-vehicle networks may have more than a hundred ECUs, these ECUs are always grouped under distinct sub-buses of a dozen or so ECUs (as already suggested by us in Figure 3 from the introduction) and the key agreement can run independently and in parallel on each of these sub-networks. This suggests the 32 ECU scenario as a realistic upper bound. We then depict

the runtime of the more efficient second protocol version in relation to both the number of ECUs and computational time between two ECUs, i.e., $\mathbf{T}_{\text{kex}}$, considering the best $24ms$ runtime between two Infineon TC297 and the slower $60.24ms$ between two TC224 (iii). Even in this worst case, the protocol runtime for 32 ECUs is only around $0.6s$. Finally, for the main scheme, we compare the influence of the bus type CAN vs. CAN-FD on the overall protocol runtime (iv) and then we depict the bus time alone in case of CAN vs. CAN-FD (v). Indeed, with CAN-FD the bus time is several times shorter but this represents less than 0.2 seconds from the overall protocol runtime of 1.1 seconds (in the worst case of the main scheme for 32 ECUs). The bus time for CAN was computed synthetically (our experiments were performed on the newer CAN-FD) by considering that a frame with a 29-bit identifier requires on average $140\mu s$ to be transmitted on a 1Mbps CAN (the maximum achievable data rate on CAN buses). Therefore $\mathbf{T}_{\text{snd32}} = 4 \times 140 = 560\mu s$ and $\mathbf{T}_{\text{snd64}} = 8 \times 140 = 1120\mu s$ which will make the transmission 2-4 times slower on CAN. With the orchestration message added, each two-party key exchange requires 5 frames on CAN-FD or 32 frames on CAN. To generalize, for a group of n ECUs, the three protocol versions require $10n$, $5n$ and $7n$ frames respectively on CAN-FD. On CAN, this extends to $64n$, $32n$ and $44n$ frames. Since a CAN bus would usually have a dozen ECUs or so, the most expensive scheme would require around one hundred CAN-FD frames or several hundred CAN frames which is only a fraction of the several thousands of frames a CAN bus can handle per second. To summarize a concrete numerical figure, for 32 nodes on CAN with the least efficient version of the schemes, the computations will last less than 1 second. Bus operations call for 2016 frames while the 500kbps CAN bus could handle around 4000fps, i.e., a busload of 50% during the first second of runtime. In a more optimistic scenario with the 2Mbps CAN-FD and the most efficient protocol scheme, the computational time decreases to $0.17s$, while the bus load tops at around 16%. As the group key exchange will be done only when the car starts, it will cause no further concerns once the car is running.

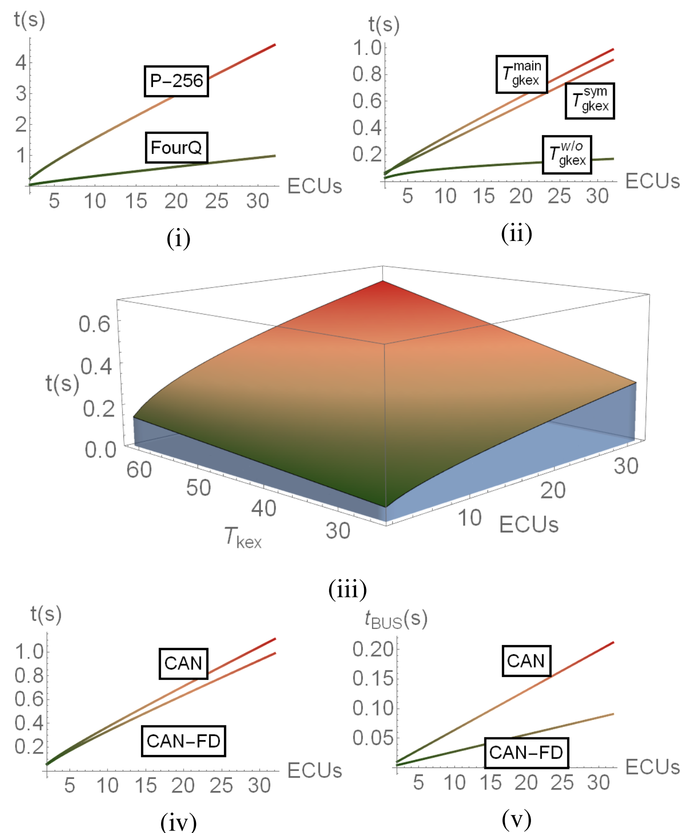


Fig. 15. Protocol runtime for $n = 2..32$ Infineon ECUs depending on: the type of curve (i), protocol version (ii), computational time (iii), bus type CAN vs. CAN-FD (iv) and bus time only CAN vs. CAN-FD (v)

VI. Conclusion

Our evaluation points out that indeed the Four$\mathbb{Q}$ elliptic curve provides the fastest results on the high-end automotive grade controller. However, this result should be interpreted with care since on the Atmel platform the NIST P-256 curve performs around two times faster than the Four$\mathbb{Q}$ curve. We believe this happens due to various optimizations of the code for ARM compilers. Still, the Four$\mathbb{Q}$-based implementation on the Infineon TC297 is the top performer. With the excellent results obtained with the Four$\mathbb{Q}$ curve, i.e., a few dozen milliseconds for a two-party key exchange on high-end cores, we show that this can be easily extended to a group key exchange between multiple nodes that costs a few hundred milliseconds depending on the number of nodes. Concretely, the fastest version of the protocols requires less than $300ms$ while the most demanding one tops at $1.1s$ in case of 32 nodes. Since the group key exchange should be performed only during vehicle start-up, or at various system resets, the evaluated group key exchange procedures should be suitable for modern vehicles and will not interfere with normal vehicle operation. Further investigations on group key exchange protocols for CAN buses are future work for us.

Acknowledgement. This work was supported by a grant of Ministry of Research and Inovation, CNCS-UEFISCDI, project number PN-III-P1-1.1-TE-2016-1317 (2018-2020).

References

[1] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham *et al.*, "Experimental security analysis of a modern automobile," in *Security and Privacy (SP), 2010 IEEE Symposium on*. IEEE, 2010, pp. 447–462.

[2] S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, K. Koscher, A. Czeskis, F. Roesner, T. Kohno *et al.*, "Comprehensive experimental analyses of automotive attack surfaces." in *USENIX Security Symposium*. San Francisco, 2011.

[3] C. Miller and C. Valasek, "A survey of remote automotive attack surfaces," *Black Hat USA*, 2014.

[4] S. Nie, L. Liu, and Y. Du, "Free-fall: Hacking tesla from wireless to can bus," *Black Hat USA*, 2017.

[5] S. Nie, L. Liu, Y. Du, and W. Zhang, "Over-the-air: How we remotely compromised the gateway, bcm, and autopilot ecus of tesla cars," *Black Hat USA*, 2018.

[6] J. Liu, S. Zhang, W. Sun, and Y. Shi, "In-vehicle network attacks and countermeasures: Challenges and future directions," *IEEE Network*, vol. 31, no. 5, 2017.

[7] C. Urquhart, X. Bellekens, C. Tachtatzis, R. Atkinson, H. Hindy, and A. Seeam, "Cyber-security internals of a skoda octavia vrs: A hands on approach," *IEEE Access*, vol. 7, pp. 146 057–146 069, 2019.

[8] C. Miller and C. Valasek, "Adventures in automotive networks and control units," *Def Con*, vol. 21, no. 260-264, pp. 15–31, 2013.

[9] *Spec. of Secure Onboard Communication*, AUTOSAR, 11 2020, r20-11.

[10] D. G. Steer, L. Strawczynski, W. Diffie, and M. Wiener, "A secure audio teleconference system," in *Proceedings on Advances in Cryptology*, ser. CRYPTO '88. Berlin, Heidelberg: Springer-Verlag, 1990, pp. 520–528.

[11] M. Steiner, G. Tsudik, and M. Waidner, "Key agreement in dynamic peer groups," *IEEE Transactions on Parallel and Distributed Systems*, vol. 11, no. 8, pp. 769–780, 2000.

[12] Y. Kim, A. Perrig, and G. Tsudik, "Group key agreement efficient in communication," *IEEE trans. on computers*, vol. 53, pp. 905–921, 2004.

[13] *Utilization of Crypto Services*, AUTOSAR, 11 2020, r20-11.

[14] *Requirements on Crypto Stack*, AUTOSAR, 11 2020, r20-11.

[15] A. Mueller and T. Lothspeich, "Plug-and-secure communication for CAN," *CAN Newsletter*, pp. 10–14, 2015.

[16] S. Jain and J. Guajardo, "Physical layer group key agreement for automotive controller area networks," in *International Cryptographic Hardware and Embedded Systems*. Springer, 2016, pp. 85–105.

[17] B. Groza, L. Popa, and P.-S. Murvay, "Tricks—time triggered covert key sharing for controller area networks," *IEEE Access*, vol. 7, pp. 104 294–104 307, 2019.

[18] S. Jain, Q. Wang, M. T. Arafin, and J. Guajardo, "Probing Attacks on Physical Layer Key Agreement for Automotive Controller Area Networks (Extended Version)," *arXiv preprint arXiv:1810.07305*, 2018.

[19] B. Groza and P.-S. Murvay, "Identity-based key exchange on in-vehicle networks: Can-fd & flexray," *Sensors*, vol. 19, no. 22, p. 4919, 2019.

[20] D. Wallner, E. Harder, R. Agee *et al.*, "Key management for multicast: Issues and architectures," RFC 2627, Tech. Rep., 1999.

[21] C. K. Wong, M. Gouda, and S. S. Lam, "Secure group communications using key graphs," *IEEE trans. on networking*, vol. 8, pp. 16–30, 2000.

[22] H. Harney, "Group key management protocol (gkmp) architecture," *RFC2094*, 1997.

[23] C. Costello and P. Longa, "Four$\mathbb{Q}$: Four-dimensional decompositions on a $\mathbb{Q}$-curve over the mersenne prime," in *Advances in Cryptology - ASIACRYPT 2015*, vol. 9452. Springer, 2015, pp. 214–235.

[24] R. de Clercq, L. Uhsadel, A. V. Herrewege, and I. Verbauwhede, "Ultra low-power implementation of ecc on the arm cortex-m0+," in *Proceedings of the 51st Annual Design Automation Conference*, ser. DAC '14. ACM, 2014, pp. 112:1–112:6.

[25] T. Unterluggauer and E. Wenger, "Efficient pairings and ecc for embedded systems," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2014, pp. 298–315.

[26] G. Hinterwälder, A. Moradi, M. Hutter, P. Schwabe, and C. Paar, "Full-size high-security ecc implementation on msp430 microcontrollers," in *International Conference on Cryptology and Information Security in Latin America*. Springer, 2014, pp. 31–47.

[27] M. Tausig and S. Schmidt, "Performance evaluation of cryptographic operations on a samr21-xpro board," in *Conf.: RIOT-OS Summit*, 2016.

[28] D. Zelle, C. Krauß, H. Strauß, and S. Karsten, "On using tls to secure in-vehicle networks," in *ARES '17 Proc. of the 12th International Conf. on Availability, Reliability and Security, Article No. 67*. ACM, 2017.

[29] L. Popa, B. Groza, and P.-S. Murvay, "Performance evaluation of elliptic curve libraries on automotive-grade microcontrollers," in *Proceedings of the 14th International Conference on Availability, Reliability and Security*, ser. ARES '19. Association for Computing Machinery, 2019.

[30] D. F. Aranha, C. P. L. Gouvêa, T. Markmann, R. S. Wahby, and K. Liao, "RELIC is an Efficient LIbrary for Cryptography."

[31] S. Bellovin and M. Merritt, "Encrypted key exchange: Password-based protocols secureagainst dictionary attacks," *Security and Privacy, IEEE Symposium on*, vol. 0, p. 72, 1992.

[32] D. P. Jablon, in *Proc. of IEEE 6th Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*, 1997, pp. 248–255.

[33] S. Woo, H. J. Jo, I. S. Kim, and D. H. Lee, "A Practical Security Architecture for In-Vehicle CAN-FD," *IEEE Trans. Intell. Transp. Syst.*, vol. 17, no. 8, pp. 2248–2261, Aug 2016.

[34] M. Bellare and P. Rogaway, "Entity authentication and key distribution," in *Advances in Cryptology - CRYPTO '93, 13th Annual International Cryptology Conference, Santa Barbara, California, USA, August 22-26, 1993, Proceedings*, vol. 773, 1993, pp. 232–249.

[35] T. Youn, Y. Lee, and S. Woo, "Practical sender authentication scheme for in-vehicle can with efficient key management," *IEEE Access*, vol. 8, pp. 86 836–86 849, 2020.

[36] Y. Wang, "Efficient identity-based and authenticated key agreement protocol," in *Trans. on Computational Science Xvii*, 2013, pp. 172–197.

[37] X. Cao, W. Kou, and X. Du, "A pairing-free identity-based authenticated key agreement protocol with minimal message exchanges," *Information Sciences*, vol. 180, no. 15, pp. 2895–2903, 2010.

[38] D. Püllen, N. A. Anagnostopoulos, T. Arul, and S. Katzenbeisser, "Using implicit certification to efficiently establish authenticated group keys for in-vehicle networks," in *2019 IEEE Vehicular Networking Conference (VNC)*, 2019, pp. 1–8.

[39] T. Tatara, H. Ogura, Y. Kodera, T. Kusaka, and Y. Nogami, "Updating a secret key for mac implemented on can using broadcast encryption scheme," in *2019 34th International Technical Conference on Circuits/Systems, Computers and Communications*, 2019, pp. 1–4.

[40] D. Boneh, C. Gentry, and B. Waters, "Collusion resistant broadcast encryption with short ciphertexts and private keys," in *Advances in Cryptology – CRYPTO 2005*, 2005, pp. 258–275.

[41] N. K. Giri, A. Munir, and J. Kong, "An integrated safe and secure approach for authentication and secret key establishment in automotive cyber-physical systems," in *Intelligent Computing*, 2020, pp. 545–559.

[42] P. S. Murvay and B. Groza, "Efficient physical layer key agreement for flexray networks," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 9, pp. 9767–9780, 2020.

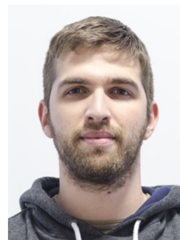
[43] S. Fassak, Y. El Hajjaji El Idrissi, N. Zahid, and M. Jedra, "A secure protocol for session keys establishment between ecus in the can bus," in *2017 International Conference on Wireless Networks and Mobile Communications (WINCOM)*, 2017, pp. 1–6.

[44] W. Diffie and M. Hellman, "New directions in cryptography," *IEEE transactions on Information Theory*, vol. 22, no. 6, pp. 644–654, 1976.

[45] "J1939-21 – Data Link Layer," SAE International, Standard, Mar. 2016.

Adrian Musuroi started his PhD studies in 2020 at Politehnica University of Timisoara (UPT). He graduated his B.Sc in 2018 and his M.Sc studies in 2020 at the same university. Since 2019 he is also working as a software developer in the automotive industry for HELLA Romania. The focus of his PhD research is on the security of automotive networks.

Bogdan Groza is Professor at Politehnica University of Timisoara (UPT). He received his Dipl.Ing. and Ph.D. degree from UPT in 2004 and 2008. In 2016 he successfully defended his habilitation thesis having as subject the design of cryptographic security for automotive networks. He has been actively involved inside UPT with the development of laboratories by Continental Automotive and Vector Informatik. He lead the CSEAMAN (2015-2017) and PRESENCE (2018-2020) projects, two national research programs dedicated to automotive security.



Lucian Popa started his PhD studies in 2018 at Politehnica University of Timisoara (UPT). He graduated his B.Sc in 2015 and his M.Sc studies in 2017 at the same university. He has a background of 4 years as a software developer and later system engineer in the automotive industry as former employee of Autoliv (2014 - 2018) and current employee of Veoneer (2018 - present). His research interests are in automotive security with focus on the security of in-vehicle buses.

Pal-Stefan Murvay is Lecturer at Politehnica University of Timisoara (UPT). He graduated his B.Sc and M.Sc studies in 2008 and 2010 respectively and received his Ph.D. degree in 2014, all from UPT. He has a 10-year background as a software developer in the automotive industry. He worked as a postdoctoral researcher in the CSEAMAN project and is currently a senior researcher in the PRESENCE project. He also leads the SEVEN project related to automotive and industrial systems security. His current research interests are in the area of automotive security.

Appendix A - Protocol procedures

This appendix describes our procedures for implementing the orchestration of the group key exchange protocol presented in section III-C. In the following algorithms we assume that the binary tree is already constructed in SoECU's memory, where each node is of type *struct ECU_t* as shown at the beginning of Algorithm 1. Using this structure, the orchestrator stores for each tree node its address, type (ECU, LECU or SoECU) and the children nodes. We also define macros for accessing each of these fields.

Algorithm 1 provides guidelines for orchestrating a group key exchange between an arbitrary number of ECUs. Here, the first procedure, i.e., *Merge-ECUs*, implements the key exchange operation between two physical or logical ECUs, where the arguments, i.e., $\mathsf{ECU_A}$ and $\mathsf{ECU_B}$, are the ECUs to be merged. In this algorithm, if both operands are *non-NULL*, a key exchange between them will be requested using their corresponding addresses. Otherwise, if only one of the operands is available (possible in the case of an odd number of ECUs on a tree level), the key exchange will be initialized using the value zero instead of the missing node's address. The effects of this command will be a direct LECU assignment, i.e., the only participant is designated as the active member and the secret value is copied to the next tree level. The second procedure, i.e., *Group-Key-Exchange*, has as single argument the root of the tree, i.e., $\mathsf{LECU_R}$ and uses a recursive postorder traversal approach in order to renew all LECU secret values. Note that renewing the root's secret value is equivalent with performing a group key exchange. For each visited LECU node, the secret random values of the children are updated first by using two recursive procedure calls. These tasks are executed in parallel and the execution continues only after they finish. Afterwards, the children nodes are merged using the previously described procedure in order to renew the secret of the tree/subtree root.

Algorithm 2 implements the ECU removal procedure that was described in section III-C. The *Remove-ECU* procedure has two arguments. The former is the tree root $\mathsf{LECU_R}$ and the latter is the address of the ECU that is required to be removed. Since every removed node must be substituted by SoECU, the procedure will set its type and address to match the orchestrator's. Then, by using a helper function, i.e., *Get-Parent*, the orchestrator will parse the tree upwards until it reaches the root. The secret share of each visited node is renewed, assuring that the removed node is excluded from all LECUs in which it was a member.

Algorithm 1 Procedures for establishing a group secret key.

```
1: #define Address(ECU)     ECU.address
2: #define Type(ECU)        ECU.type
3: #define Left(ECU)        ECU.left_child
4: #define Right(ECU)       ECU.right_child

5: enum ecu_type {
6:    ECU,
7:    LECU,
8:    SoECU
9: };

10: struct ECU_t {
11:    uint8 address;
12:    enum ecu_type type;
13:    struct ECU_t *left_child;
14:    struct ECU_t *right_child;
15: };

16: procedure MERGE-ECUS(ECU_A, ECU_B)
17:    if ECU_A ≠ NULL and ECU_B ≠ NULL then
18:       STS-KEYEXCH(Address(ECU_A), Address(ECU_B))
19:    else if ECU_A = NULL and ECU_B ≠ NULL then
20:       STS-KEYEXCH(Address(ECU_B), 0)
21:    else if ECU_B = NULL and ECU_A ≠ NULL then
22:       STS-KEYEXCH(Address(ECU_A), 0)
23:    end if
24: end procedure

25: procedure GROUP-KEY-EXCHANGE(LECU_R)
26:    if Type(LECU_R) = LECU then
27:       if Left(LECU_R) ≠ NULL then
28:          GROUP-KEY-EXCHANGE(Left(LECU_R))
29:       end if
30:       if Right(LECU_R) ≠ NULL then
31:          GROUP-KEY-EXCHANGE(Right(LECU_R))
32:       end if
33:       MERGE-ECUS(Left(LECU_R), Right(LECU_R))
34:    end if
35: end procedure
```

Algorithm 2 Procedures for removing an ECU

```
1: procedure GET-PARENT(LECU_R, addr)
2:    ECU_left ← Left(LECU_R)
3:    ECU_right ← Right(LECU_R)
4:    if ECU_left ≠ NULL then
5:       if Address(ECU_left) = addr then
6:          Parent ← LECU_R
7:       else
8:          GET-PARENT(ECU_left, addr)
9:       end if
10:   end if
11:   if ECU_right ≠ NULL then
12:      if Address(ECU_right) = addr then
13:         Parent ← LECU_R
14:      else
15:         GET-PARENT(ECU_right, addr)
16:      end if
17:   end if
18: end procedure

19: procedure REMOVE-ECU(LECU_R, ECU)
20:    addr ← Address(ECU)
21:    Type(ECU) ← SoECU
22:    Address(ECU) ← Address(SoECU)
23:    GET-PARENT(LECU_R, addr)
24:    do
25:       MERGE-ECUS(Left(Parent), Right(Parent))
26:       addr ← Address(Parent)
27:       GET-PARENT(LECU_R, addr)
28:    while Parent ≠ LECU_R
29: end procedure
```