Session Key Distribution Made Practical for CAN and CAN-FD Message Authentication

Yang Xiao
Virginia Tech
xiaoy@vt.edu

Shanghao Shi
Virginia Tech
shanghaos@vt.edu

Ning Zhang
Washington University in St. Louis
zhang.ning@wustl.edu

Wenjing Lou
Virginia Tech
wjlou@vt.edu

Y. Thomas Hou
Virginia Tech
thou@vt.edu

ABSTRACT

Automotive communication networks, represented by the CAN bus, are acclaimed for enabling real-time communication between vehicular ECUs but also criticized for their lack of effective security mechanisms. Various attacks have demonstrated that this security deficit renders a vehicle vulnerable to adversarial control that jeopardizes passenger safety. A recent standardization effort led by AUTOSAR has provided general guidelines for developing next-generation automotive communication technologies with built-in security mechanisms. A key security mechanism is message authentication between ECUs for countering message spoofing and replay attack. While many message authentication schemes have been proposed by previous work, the important issue of session key establishment with AUTOSAR compliance was not well addressed. In this paper, we fill this gap by proposing an AUTOSAR-compliant key management architecture that takes into account practical requirements imposed by the automotive environment. Based on this architecture, we describe a baseline session key distribution protocol called SKDC that realizes all designed security functionalities, and propose a novel secret-sharing-based protocol called SSKT that yields improved communication efficiency. Both SKDC and SSKT are customized for CAN/CAN-FD bus deployment. We implemented the two protocols on commercial microcontroller boards and evaluated their performance with hardware experiment and extrapolation analysis. The result shows while both protocols are performant, SSKT achieves superior computation and communication efficiency at scale.

CCS CONCEPTS

• **Security and privacy** → **Key management**; **Security protocols**; *Embedded systems security.*

KEYWORDS

Automotive communication networks, CAN, CAN-FD, message authentication, key distribution, secret sharing

ACM Reference Format:
Yang Xiao, Shanghao Shi, Ning Zhang, Wenjing Lou, and Y. Thomas Hou. 2020. Session Key Distribution Made Practical for CAN and CAN-FD Message Authentication. In *Annual Computer Security Applications Conference (ACSAC 2020), December 7–11, 2020, Austin, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3427228.3427278

1 INTRODUCTION

Modern vehicles rely on a fleet of on-board microcontroller modules, known as Electronic Control Units (ECUs), for sensor data processing and real-time subsystem control. These ECUs, connected by one or more automotive communication networks, exchange control information with each other through standardized communication protocols. The Controller Area Network protocol [31], commonly known as the CAN bus protocol, is the de facto communication standard for safety-critical subsystems in a modern vehicle. CAN-FD [32] is the official extension of CAN which supports higher transmission bit rate and longer data payloads.

These automotive communication protocols are designed with the aim of optimizing communication efficiency, but often lack modern security protections such as message authentication. Take CAN for example, it adopts a broadcast-and-subscribe messaging paradigm wherein messages carry no sender or receiver information except for the message ID. Any message sent by any ECU reaches all ECUs on the bus. While an ECU is expected to only read the messages it subscribes, due to the lack of authentication of message source or receiver, a malicious ECU can legally read all messages and inject new messages with arbitrary ID and payload. Various attacks have demonstrated that this vulnerability can be exploited for malicious control of the vehicle behavior [19, 25] and reverse engineering of ECU messaging patterns [20, 29, 42]. These attacks are generally achieved by injecting malicious messages and analyzing existing messages through the OBD-II port or an exposed bus interface.

Recognizing the pressing need for securing the control network, industry-wide standardization effort, led by the AUTOSAR consortium, has provided architectural guidelines for in-vehicle security mechanisms that counter the message-related attacks. The AUTOSAR specification of Secure Onboard Communication (SecOC) [2] suggests mandatory message authentication, as well as the using message counter for replay attack resistance. Specially, it stipulates that message authentication shall be performed on group basis, i.e., all subscribing nodes of a message ID need to share a secret key which is used for the authentication of that message ID. This group-based authentication paradigm guarantees a practical level of security and also preserves the efficiency of group messaging associated with the broadcast nature of in-vehicle buses.

However, key establishment in AUTOSAR SecOC receives little attention. Two well-received AUTOSAR-compliant CAN message authentication schemes [27, 30] assume that a long-term secret key for each CAN message ID has been pre-distributed to the tamper-proof memory region of the subscribing ECUs. [27] directly applies the long-term keys to message authentication, while [30] further configures that *session keys* shall be derived from the long-term keys and used only in a short term (eg., vehicle start-up to shutdown). Using session keys is a good security practice for enabling replay attack resistance and forward secrecy. However, the session key derivation mechanism in [30] requires bulk storage of message-specific long-term keys in ECU's tamper-proof memory, which can be a constraining factor when the number of message IDs is large. In comparison, [38] adopts centralized key management in that each ECU only needs to share one key-encryption key with the trusted key server, and session keys for group message authentication are distributed by the key server to ECUs in a key distribution center (KDC) style. Centralized key management is a practical choice for in-vehicle networks, wherein the connected ECUs are static in place and limited in quantity. Nonetheless, the KDC-style key establishment requires key distribution to be done one ECU at a time, the communication overhead thereof can be significant for large bus networks.

Our Contribution We tackle the problem of efficient session key establishment for message authentication in automotive communication networks. We identify four practical requirements for key establishment in automotive environment: 1) lightweight cryptography, 2) communication efficiency, 3) AUTOSAR-compliant security, and 4) flexibility with on-demand ECUs. To fulfill these goals, we formulate a centralized key management architecture and opt for key distribution (in contrast to key agreement), which is inline with prior wisdom [38, 43] that adopted centralized key management for efficiency purposes. Specifically, we introduce a central key server (KS) for managing the system epoch counter and the generation of session keys for different messages. KS shares a long-term secret key with each ECU called ECU key, which will be used in the distribution of session keys. KS also accommodates the session key requests from on-demand ECUs when they switch on during normal vehicle operation.

Based on this architecture, we focus on CAN and CAN-FD bus and construct session key distribution protocols in two steps. First, we describe a baseline KDC-style protocol, dubbed SKDC, wherein KS uses ECU keys to encrypt a session key and delivers it to the corresponding ECUs one by one. We take into account the nuances in CAN and CAN-FD frame structure and key confirmation. Afterwards, we propose a new session key distribution protocol, named SSKT, which achieves the same functionalities while yielding significantly improved communication efficiency. SSKT employs an adapted version of the secret-sharing-based key transfer primitive (SKT) [13] for the delivery of session keys. SSKT takes advantage of the broadcast nature of the CAN/CAN-FD bus in that all ECUs receive the materials for recovering a subscribed message session key from KS at the same time and start recovering this key simultaneously. Specially, the key recovery procedure relies on polynomial interpolation in a finite field, wherein the most time-consuming steps—computing Lagrangian coefficients—can be pre-computed for achieving optimal time efficiency.

We provide a proof-of-concept implementation of the proposed SKDC and SSKT protocols on commercial microcontroller boards commonly used for vehicular ECU emulation. We built a CAN bus test platform for validation and feasibility evaluation and performed extrapolated analyses with performance benchmarks. The result shows that SSKT achieves better computation and communication efficiency at scale, at the cost of increased memory footprint.

In summary, we make the following contributions:

- We propose a practical key management architecture for message authentication in automotive communication network which is compliant with AUTOSAR specification.
- Based on the architecture, we propose a baseline KDC-style key distribution protocol called SKDC, and a secret-sharing-based protocol called SSKT which yields improved communication efficiency. Both protocols are customized for session key establishment in CAN and CAN-FD bus.
- We provide an proof-of-concept implementation of SKDC and SSKT and evaluate their performance with hardware experiment and extrapolation analysis. The result demonstrates the feasibility of both protocols as well as SSKT's advantage in computation and communication efficiency at scale.

2 BACKGROUND

2.1 Automotive Communication Network

Modern automobile development has seen the increasing complexity of in-vehicle electrical and electronic systems. A passenger car nowadays has up to 80 electronic control units (ECU) connected by one or more automotive communication networks [9]. An automotive communication network typically consists of three layers of components: the application-specific ECU, the data link layer, and the physical layer [37]. The data link layer specifies the frame structure and translates ECU messages into logical bits. It also encodes certain message processing rules that essentially realize a communication protocol. The physical layer specifies the underlying wire technology and also includes an medium access unit that converts logical bits into physical signals, and vice versa.

Widely used automotive communication networks include the Controller Arean Network (CAN) [31], Local Interconnect Network (LIN) [7], FlexRay [24], and Media Oriented Systems Transport (MOST) [8]. They are purposed for different in-vehicle electronic systems and utilize different physical wire technologies. They are all bus networks and bridged by a central gateway that also serves as a message outlet for vehicle diagnostics [44]. In this work we address the security issues of CAN and its extension CAN-FD, as currently most safety-critical ECUs such as engine, transmission, and ABS control units are connected by a CAN bus [19].

2.2 CAN and CAN-FD Basics

CAN was proposed by Robert Bosch GmbH in 1986 as an automotive communication bus, with the latest version (2.0) released in 1991 [31]. CAN-FD, wherein FD stands for Flexible Data-rate, was released by Bosch in 2012 as a CAN extension [32]. Bosch's specification on CAN and CAN-FD are standardized in ISO 11898-1 as data link layer protocols [17], while the physical layer is independently standardized in ISO 11898-2/3 [16, 18].

Frame Format ECUs communicate via message frames. There are four types of message frames, namely data frame, remote frame, error frame, and overload frame, of which the data frame is the default type for data transmission. The data frame formats of CAN and CAN-FD are illustrated in Figure 1. It should be noted that message frames carry no sender or receiver information, while the base ID and extended ID fields comprise of the message identifier. One major difference between a CAN data frame and a CAN-FD data frame lies in the maximum size of data payload. CAN supports up to 8 bytes of data and while CAN-FD supports up to 64 bytes. This extended payload capacity allows a CAN-FD data frame to better integrate standard cryptographic functions.

Broadcast Bus Any message sent is broadcast to the bus. Each ECU keeps a message subscription list on what messages to read, enforced by a filter and mask mechanism on the message identifier. Only one message can be broadcast at a time, during which all ECUs are synchronized to receive every bit. One key difference between CAN and CAN-FD is that the latter's medium access unit allows an adjustable bit rate for the transceiving of the DLC, data and CRC fields; bit rate for other fields is consistent with the coexisting CAN protocol. This makes CAN-FD messages backward-compatible with CAN while enabling faster transmission of useful data.

Message Arbitration The collision of messages are detected and resolved through the message arbitration mechanism. The arbitration field contains the message ID, which is binary-encoded in a way that a message of higher priority has a lower ID value. The logic bit "0" and "1" are defined as "dominant" and "recessive", corresponding to high and low voltages in the physical layer respectively. The start-of-frame bit (SOF) is defined as "0" and after which all ECUs are synchronized for bit-wise transmission and detection on the bus. As a result, a message frame with a lower message ID has more preceding logic zeros and automatically overrides messages with higher message IDs. ECUs that have detected the transmission of a higher-priority message frame during message arbitration will wait until the current frame finishes before sending its message frame.

2.3 CAN and CAN-FD Message Authentication

The current CAN and CAN-FD specifications have limited security measures. Due to the broadcast nature of the bus network and frame format, a receiver ECU needs to pick messages based on message IDs, while with no assurance on authenticity of the sender or integrity of the message. An adversary can read all messages and inject arbitrary messages through exposed bus interfaces, such as the OBD-II port or a compromised wired/wireless interface of an ECU. This vulnerability allows attackers to maliciously control of the vehicle behavior [19, 25] and reverse-engineer ECUs [20, 29, 42].

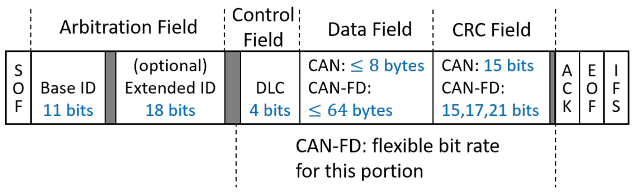

Arbitration Field
Control Field
Data Field
CRC Field
SOF
Base ID 11 bits
(optional) Extended ID 18 bits
DLC 4 bits
CAN: ≤ 8 bytes CAN-FD: ≤ 64 bytes
CAN: 15 bits CAN-FD: 15,17,21 bits
ACK
EOF
IFS
CAN-FD: flexible bit rate for this portion

Figure 1: Data frame formats of CAN and CAN-FD.

AUTOSAR Specification To provide better security in automotive environment, the industry-wide consortium AUTOSAR recommends using authentication mechanisms for secure communication. The AUTOSAR SecOC specification [2] defines two authentication functionalities for normal message-based communication: entity authentication and message authentication. The former validates the sender identity of the ensuing communication, while the latter ensures the sender of a message is entitled to send this message (i.e., source authenticity) as well as message integrity. For current automotive communication buses, entity authentication would require pairwise ECU secret keys or a dedicated authentication server node, while message authentication can be conveniently integrated into regular messaging with limited cryptographic overhead. Specifically, AUTOSAR SecOC provides three practical guidelines for message authentication: 1) Authentication is associated with message groups. That is, a message authentication code (MAC) of a regular message proves the message sender is a valid subscriber of the message ID. 2) A counter is included in MAC computation for message freshness and replay attack resistance. 3) For cryptographic strength, 128-bit secret keys and MACs of at least 64 bits are recommended.

Several AUTOSAR-compliant message authentication schemes have been proposed for CAN bus [27, 30, 38]. Due to the 64-bit payload limit of a CAN data frame, they all resort to sending the MAC of a message in a subsequent data frame, though they differ in the choice of MAC algorithm and placement of message counters. There were also numerous CAN message authentication schemes proposed prior to AUTOSAR. These works contain valuable lessons but are not fully compliant with the three practical guidelines provided by AUTOSAR SecOC. Interested readers are referred to §9 for a brief review on these legacy schemes.

2.4 Message Authentication Key Management

Noticeably missing from AUTOSAR specification is the establishment of authentication keys. vatiCAN [27], the first known AUTOSAR-compliant CAN message authentication mechanism, assumes the message-specific keys are pre-distributed to ECUs during vehicle assembly. These keys are used for message authentication throughout the vehicle's lifetime. However, using long-term keys directly for authentication is not a good security practice, since the possible compromise of a long-term authentication key will jeopardize the operation of all ECUs that hold this key. Moreover, key storage can be challenging if a resource-limited vehicular ECU needs to subscribe to a large amount of message IDs.

Using Session Keys A more secure practice is to dynamically generate keys for message authentication for every new communication session. These *session keys* need to be established in ECUs according to their message subscription profiles. Legacy schemes such as CANAuth [39], LiBra-CAN [11], and MaCAN [14] have adopted session keys, though they do not conform to AUTOSAR's guidance on message ID-based grouping. LeiA [30], the first AUTOSAR-compliant proposal that adopts session keys, assumes message-specific long-term keys are pre-distributed to ECUs (same as vatiCAN), which together with the system epoch number are used for deriving short-term session keys for every vehicle start-up. While LeiA's session key derivation mechanism yields minimal communication and computation overhead, it risks the same long-term key compromise and storage problem as vatiCAN does since each ECU stores the long-term key for every subscribed message ID.

Alternatively, session keys can be distributed in a centralized manner. VulCAN [38] employs a *key server* to generate session keys and maintain the system epoch in a KDC-style. The key server and each ECU share a unique long-term key-encryption key (KEK). Every session key is then delivered to the subscribing ECUs separately. Compared to LeiA, this centralized scheme saves the precious tamper-resistant memory space at every ECU. It also enables the implementation of other useful security mechanisms, including key confirmation and ECU entity authentication, which can be used for system liveness detection. Partially for this reason, our proposed system architecture also adopts the centralized paradigm.

2.5 Secret-sharing-based Key Establishment

Alternative to encryption-based methods, secret key can also be delivered via secret sharing. Here we are interested in one particular paradigm of secret sharing called (n, t)-threshold scheme: a dealer splits the secret into n shares and gives one share to each of the n participants. Any combination of t or more shares can be used to reconstruct the secret.

The SKT Primitive The secret sharing-based key transfer primitive (SKT) was proposed by Harn and Lin [13] in 2010 and is built upon Shamir's secret sharing [34]. SKT assumes t members wish to establish a group key with the help of a key generation center (KGC). KGC shares a long-term secret pair (x_i, y_i) with each member i. In the beginning, KGC generates a secret key sk and each member i sends a random offset R_i to KGC. KGC generates a degree-t polynomial $f(x)$ using the $t+1$ points: $\{(x_l, y_l \oplus R_l)\}_{l \in [t]}$ and $(0, sk)$, wherein $\oplus$ means XOR. KGC then picks t auxiliary random points $\{P_l\}_{l \in [t]}$ on $f(x)$ (different from any (x_i, y_i) or $(0, sk)$) and broadcasts them to group members. Group member i recovers the polynomial $f(x)$ and the group secret key sk via polynomial interpolation on the $t+1$ points (aka. secret shares): the public $\{P_l\}_{l \in [t]}$ and the confidential $(x_i, y_i \oplus R_i)$. All arithmetic operations are performed in $\mathcal{Z}_w^*$ (i.e., modulo w). Compared to KDC-style schemes, SKT features an advantage in communication efficiency: KGC only needs to broadcast the auxiliary points $\{P_l\}_{l \in [t]}$ once and all members in the designated group can recover the group key simultaneously.

We note that there are other secret-sharing-based schemes for key establishment. Readers are referred to §9 for a brief review.

3 SYSTEM ARCHITECTURE

3.1 Network Model

We consider a CAN/CAN-FD bus network of N ECUs with M message identifiers that need authentication service. We assume $N < 2^8$ and $M < 2^{18}$ so that the Base ID and Extended ID fields of a CAN or CAN-FD data frame can provide sufficient space to encode ECU ID and message ID respectively. Every ECU is preassigned an ECU ID $EID_i \in \{0, 1\}^8, \forall i \in [N]$ ($[N]$ denotes $\{1, 2, ..., N\}$). Every message identifier that needs authentication service is preassigned a message ID for key assignment purposes: $MID_j \in \{0, 1\}^{18}, \forall i \in [M]$. Each ECU i has a determined subscription list $\mathbf{SL}_i$, which contains MIDs of messages it will send and receive with MAC.

System Goal The goal is to establish session keys for message authentication in ECUs at the start of every automotive communication session. We define a *communication session* as the period from vehicle start-up to shut-off, in consistency with [27, 30, 38]. For every new session, each MID_j is assigned a random 128-bit session key sk_j for message authentication. All ECUs that subscribe to a common MID need to get a session key for that MID. In other words, for any (a, b, j) tuple that satisfies $MID_j \in \mathbf{SL}_a \cap \mathbf{SL}_b$, ECU a and b need to share the session key sk_j.

Threat Model The adversary eavesdrops all CAN/CAN-FD messages transmitted in the bus and can inject messages with arbitrary message ID and payload. The adversary can replay any previously eavesdropped message. The adversarial goal is to acquire the session keys for the subsequent authenticated communication. And in the case that an ECU is compromised, the adversary can only obtain the victim's session keys for the current session.

3.2 Practical Considerations on Session Key Establishment

In addition to the basic functionality and compatibility with CAN/CAN-FD bus, we identify four practical requirements for a session key establishment protocol to be deployed in automotive environments:

- **R1: Lightweight Cryptography** Vehicular ECUs are generally resource-constrained. The computational burden of each ECU should be limited.
- **R2: Communication Efficiency** Due to the underlying bus network and the arbitration process of CAN/CAN-FD, only one message can be broadcast at a time. The overall message complexity should be controlled so the protocol finishes in a short time.
- **R3: AUTOSAR-compliant Security** The protocol should use 128-bit secret keys and 64-bit MACs as are recommend by AUTOSAR SecOC [2]. It should be resistant to replay attacks in that an incremental system epoch shall be instantiated at all ECUs for every new session.
- **R4: Flexibility with On-demand ECU** Not all ECUs are on at vehicle start-up. ECUs that switch on on-demand during driving should obtain its subscribed sessions swiftly with minimal impact on normal operation.

We remark that **R1** effectively rules out public key cryptography, which involves long-term storage of lengthy asymmetric keys and cumbersome computation at ECUs. Meanwhile, the straightforward key derivation method adopted by LeiA [30] is also not favorable

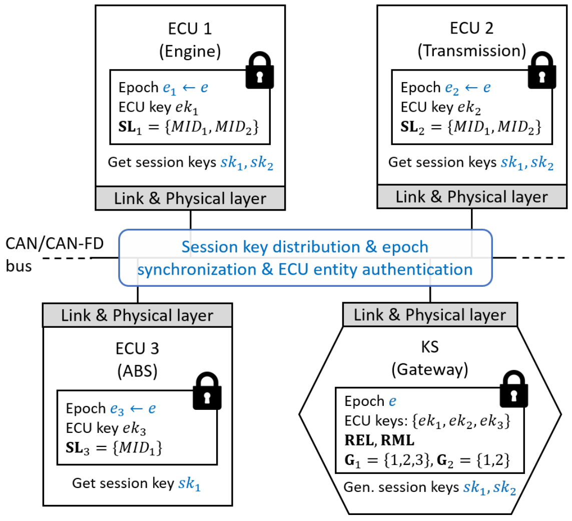

ECU 1
(Engine)
Epoch $e_1 \leftarrow e$
ECU key ek_1
$\mathbf{SL}_1 = \{MID_1, MID_2\}$
Get session keys sk_1, sk_2
Link & Physical layer
ECU 2
(Transmission)
Epoch $e_2 \leftarrow e$
ECU key ek_2
$\mathbf{SL}_2 = \{MID_1, MID_2\}$
Get session keys sk_1, sk_2
Link & Physical layer
CAN/CAN-FD bus
Session key distribution & epoch synchronization & ECU entity authentication
Link & Physical layer
ECU 3
(ABS)
Epoch $e_3 \leftarrow e$
ECU key ek_3
$\mathbf{SL}_3 = \{MID_1\}$
Get session key sk_1
Link & Physical layer
KS
(Gateway)
Epoch e
ECU keys: $\{ek_1, ek_2, ek_3\}$
REL, **RML**
$\mathbf{G}_1 = \{1,2,3\}, \mathbf{G}_2 = \{1,2\}$
Gen. session keys sk_1, sk_2

Figure 2: An example of the proposed system architecture. ECU 1, 2, 3 **subscribing** MID_1 **and ECU** 1, 2 **subscribing** MID_2**. ECU** 1, 2, 3 **all switch on from the session onset.**

since it requires the long-term storage of various message-specific master keys. **R2** rules out classical key agreement protocols which typically yield $O(N^2)$ message complexity. The system epoch value imposed by **R3** can be used in the ensuing message authentication as the session indicator. **R3** and **R4** entail a trusted entity to manage the system epoch value and keep records of the on-demand ECUs. Moreover, safety-critical vehicle operations should be blocked until session keys are established. For gasoline and diesel vehicles, we recommend the establishment of session keys be completed before engine ignition. For electric vehicles, we recommend it be done before unblocking vehicle motion. For these reasons, we consider centralized *key distribution* by a trusted entity the best option for session key establishment in automotive communication networks.

3.3 A Key Server-based Architecture

We introduce a trusted entity called *Key Server* (KS) to generate and distribute symmetric session keys to ECUs in a centralized manner. The inclusion of KS is a practical design choice for CAN and CAN-FD buses for two reasons. First, only one message can be broadcast at a time in the bus, and all ECUs are synchronized to bits during message arbitration. This allows one entity to distribute information in an publicly observable manner. Second, vehicular ECUs have fixed quantity and are placed in fixed locations, which provides convenience for centralized management such as key distribution and ECU identity authentication. In practice, KS can be instantiated on an existing ECU, such as the gateway ECU enhanced with a powerful processor. We also assume KS can block engine ignition and vehicle motion until session key distribution is completed.

Figure 2 shows an example of the proposed system architecture. Each ECU i shares a 128-bit long-term ECU key ek_i and its subscription list $\mathbf{SL}_i$ with KS, which are stored in tamper-resistant memory regions. ek_i's are used in the session key distribution protocol as well as optional ECU entity authentication. Epochs are used for resisting replay attacks, in accordance to AUTOSAR specification. Specially, a 64-bit system epoch e is managed by KS. We assume KS tracks the highest local epoch e_i of any ECU i it has observed during normal communication, and keeps the system epoch e updated to such e_i. For a new session, KS increments e and starts the session key distribution protocol. e should be included in every ensuing protocol message. All ECUs should synchronize local epoch to e, while individual e_i may diverge during the ensuing normal communication [30, 38]. Moreover, KS stores a required ECU list (**REL**) and a required message list (**RML**). **REL** specifies the ECUs that require message authentication from session onset (in contrast to on-demand ECUs). Correspondingly, **RML** specifies the message IDs that require the assignment of session keys from session onset. We further define $\mathbf{G}_j$ as the group of ECUs that subscribe to MID_j and are required to start from the onset. That is, $\mathbf{G}_j = \{\forall i | MID_j \in \mathbf{SL}_i \text{ and } i \in \mathbf{REL}\}$. The size of $\mathbf{G}_j$ is denoted t_j.

Based on the KS-based architecture, we proceed to construct the session key distribution protocol for CAN and CAN-FD bus. The baseline, named SKDC, is a KDC-style protocol which realizes functionalities including session key distribution and confirmation. The second protocol, named SSKT, achieves the same functionalities but yields higher communication efficiency.

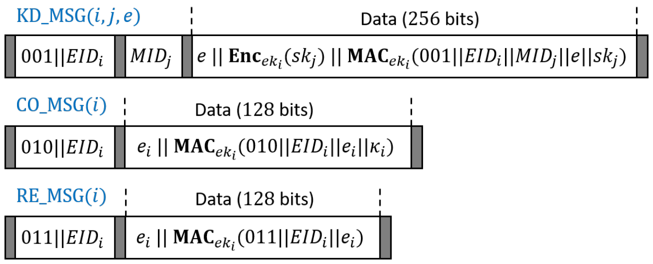
4 BASELINE: THE SKDC PROTOCOL

A classical KDC relies on pre-shared *key-encryption keys* (KEK) to distribute temporary keys in encrypted channel. In SKDC, KS serves the KDC role and uses the pre-shared long-term ECU keys as KEKs. To distribute session key sk_j, for each ECU i that subscribes MID_j, KS encrypts sk_j with ek_i and delivers it to ECU i. To fit into CAN/CAN-FD's message framework, we define three special protocol messages: *key delivery* KD_MSG, *confirmation* CO_MSG, and *request message* RE_MSG, with formats specified in Figure 3. $\mathbf{MAC}_{ek_i}(\cdot)$ is a 64-bit truncated MAC and can be derived using standard HMAC or keyed hash. The encryption function $\mathbf{Enc}_{ek_i}(\cdot)$ can be a standard symmetric-key encryption scheme such as AES.

4.1 Protocol Workflow

For a new session, the SKDC protocol proceeds as follows:

- **Phase 1: Key Generation** KS generates random 128-bit session keys for all M message IDs: $\{sk_1, sk_2, ..., sk_M\}$ and increments e.
- **Phase 2-1: Key Delivery** For every message $j \in \mathbf{RML}$ and each ECU $i \in \mathbf{G}_j$, KS sends out KD_MSG(i, j, e) onto the bus. The message payload contains the epoch e, ciphertext of session key sk_j encrypted with ek_i, as well as a MAC for the arbitration field and the session key sk_j, which will be used by ECU i for authenticity and integrity check.
- **Phase 2-2: Key Recovery** When ECU i receives message KD_MSG(i, j, e) from the bus and $j \in \mathbf{SL}_i$, it obtains the epoch e and verifies $e > e_i$ if for the first time or $e = e_i$ otherwise. It then decrypts sk_j with ek_i. After validating the received information with the MAC, ECU i accepts sk_j as the session key for MID_j. It updates e_i to e if $e > e_i$. If validation fails, ECU i sends out a RE_MSG to request KS to resend the KD_MSGs.



$\text{KD\_MSG}(i, j, e)$
Data (256 bits)
$001||EID_i$
MID_j
$e \,||\, \mathbf{Enc}_{ek_i}(sk_j) \,||\, \mathbf{MAC}_{ek_i}(001||EID_i||MID_j||e||sk_j)$
$\text{CO\_MSG}(i)$
Data (128 bits)
$010||EID_i$
$e_i \,||\, \mathbf{MAC}_{ek_i}(010||EID_i||e_i||\kappa_i)$
$\text{RE\_MSG}(i)$
Data (128 bits)
$011||EID_i$
$e_i \,||\, \mathbf{MAC}_{ek_i}(011||EID_i||e_i)$

Figure 3: Protocol message formats of SKDC.

- **Phase 3: Confirmation** Once ECU i obtains all subscribed session keys specified in SL_i, it proceeds to generate a ek_i-keyed MAC as a digest of the session keys. In Figure 3 we use κ_i to denote the concatenation of the session keys. The MAC is sent out via CO_MSG(i).
  When KS receives CO_MSG(i), it verifies the received $e_i = e$ and validates e_i and the session keys with the MAC. (If multiple ECUs attempt to send CO_MSG at the same time, the order is regulated by the arbitration process, i.e., ECU with the lowest EID gets to send first.)

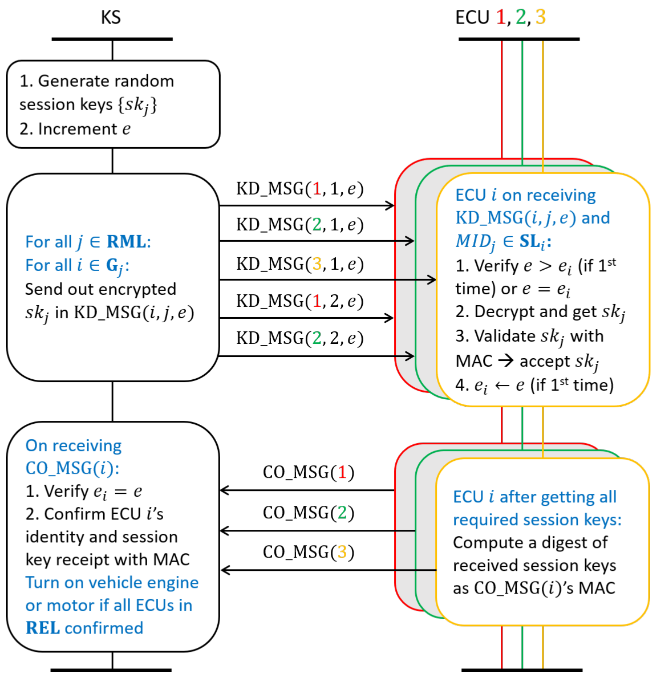
We remark that a CO_MSG also fulfills ECU entity authentication. An execution example of SKDC in illustrated in Figure 4. There are several practical considerations for deploying SKDC in CAN and CAN-FD buses. For CAN-only bus, the payload of KD_MSG and CO_MSG will exceed the 64-bit limit and thus needs to be transmitted in break-out messages. According to previous CAN authentication schemes [27, 30, 38], the break-out messages can employ a special encoding rule in the arbitration field to maintain a consistent priority level, for instance, each subsequently increments by one. Moreover, to accelerate overall protocol execution, and since we assume KS has a powerful processor which is significantly faster than normal ECU processors, KS can pre-compute the key digests as well as MACs of the CO_MSGs that it expects to receive right after sending out all KD_MSGs.

For an on-demand ECU that does not start from the session onset but switches on during normal operation, it sends out a RE_MSG and expect KD_MSGs from KS, then performs the same Phase-2 operations described in the workflow. Confirmation is not required for on-demand ECUs as the vehicle is already in operation mode.

4.2 Security Analysis

We analyze the security of SKDC from three aspects: session key correctness, session key and ECU key confidentiality, ECU entity authentication and liveness.

First, session key correctness means the session key received by an ECU is indeed the one generated by KS for the current session. This is equivalent to the authenticity and integrity of KD_MSG(i, j, e), which contains a MAC computed with ECU key ek_i only known to KS and ECU i. Without knowing ek_i, the attacker needs to forge the message by coinciding a MAC and send it to the ECU, which yields expected 2^{63} guesses to succeed. And this should be done before the ECU receives the authentic KD_MSG(i, j, e) from KS. The incremental epoch value ensures that the attacker needs to start over for every new session, fulfilling replay attack resistance.



KS
ECU 1, 2, 3
1. Generate random session keys $\{sk_j\}$
2. Increment e
For all $j \in$ **RML**:
For all $i \in \mathbf{G}_j$:
Send out encrypted sk_j in KD_MSG(i, j, e)
KD_MSG(1, 1, e)
KD_MSG(2, 1, e)
KD_MSG(3, 1, e)
KD_MSG(1, 2, e)
KD_MSG(2, 2, e)
ECU i on receiving KD_MSG(i, j, e) and $MID_j \in \mathbf{SL}_i$:
1. Verify $e > e_i$ (if 1st time) or $e = e_i$
2. Decrypt and get sk_j
3. Validate sk_j with MAC → accept sk_j
4. $e_i \leftarrow e$ (if 1st time)
On receiving CO_MSG(i):
1. Verify $e_i = e$
2. Confirm ECU i's identity and session key receipt with MAC
Turn on vehicle engine or motor if all ECUs in **REL** confirmed
CO_MSG(1)
CO_MSG(2)
CO_MSG(3)
ECU i after getting all required session keys:
Compute a digest of received session keys as CO_MSG(i)'s MAC

Figure 4: Example workflow of SKDC. Message subscription follows Fig. 2.

Second, since session key sk_j is encrypted with the ECU key ek_i in KD_MSG(i, j, e), its confidentiality is equivalent to that of ek_i. We note that in CO_MSG(i) and RE_MSG(i), ECU key ek_i is used to generate the MAC. To uncover ek_i, the attacker needs to target a specific RE_MSG(i) (or a CO_MSG(i) in the unlikely case the attacker compromised ECU i's session keys) and brute-force the key universe until a correct MAC is found. If we model the MAC algorithm as a random oracle that takes ECU key as input (e.g., ek_i-keyed Hash MAC), the attacker would expect 2^{127} MAC evaluations to succeed. This is commonly believed to be computationally infeasible for practical attackers.

Lastly, the MAC in CO_MSG(i) allows KS to verify ECU i's identity and liveness. The validation of CO_MSG(i)'s MAC confirms ECU i's receipt of its subscribed session keys. The confirmation step essentially fulfills ECU entity authentication and liveness detection, and the security thereof reduces to the security of the MAC algorithm and confidentiality of ECU keys. If KS does not receive a valid CO_MSG(i) for the current session, it will continue blocking vehicle motion. For on-demand ECUs, their entity and liveness are proven by the MAC of a RE_MSG.

5 THE SSKT PROTOCOL

The baseline protocol is straightforward and can be implemented with off-the-shelf cryptographic libraries. However, it faces a performance challenge: the same session key has to be delivered to one ECU at a time, accumulating significant communication overhead. In comparison, secret sharing-based schemes, epitomized by the SKT primitive [13] as we introduced in §2.5, provide a communication-efficient alternative. Next, we construct the SSKT protocol based on a customized version of SKT which achieves the same functionalities of SKDC but yields higher communication efficiency.

5.1 SKT Limitations

While we intend to harness its advantage in communication efficiency for session key distribution, the original SKT can not be directly applied to automotive settings due to the following facts:

- **SKT-F1** An potential attack scenario called insider spoofing may result in the compromise of the spoofed victim's long-term secret pair [13]. Specifically, the attacker, also a group member, forges and sends out the same random number for the victim member in two protocol sessions. The difference between the two polynomials may help the attacker deduce the victim's long-term secret pair. To prevent this attack, SKT uses the product of two large primes as the modulus w and relies on the intractability of large number factorization for attack prevention.
- **SKT-F2** One SKT instance establishes one group key. If there are multiple groups among all participants, independent SKT instances are needed to assign a key for each group. For a participant that subscribes to multiple groups, it needs to upload a new random number R_i to KGC in every instance.

SKT-F1 implies that the modulus w should be at least a thousand bits long to provide practical security. Modular arithmetic at this scale has far exceeded the payload capacity of CAN/CAN-FD data frames and processing capability of commercial ECUs. Setting in our architecture, **SKT-F2** would require an ECU to send an updated random number for each session key it subscribes, leading to excessive communication and processing overhead.

5.2 SSKT: Optimized for Automotive Deployment

To fit in the proposed architecture in Fig. 2, in the proposed SSKT protocol we let KS assume the role of the KGC and ECUs the group members. The pre-shared long-term ECU key ek_i takes the pair form $(x_i, y_i), i \in [N]$. x_i, y_i are 128-bit secret keys. To address the challenges raised by SKT facts and to better accommodate automotive environments, we make the following incremental improvements to SKT in SSKT.

First, as an alternative strategy of preventing insider spoofing attack, we let KS generate a random number R_i once per session and send it to the respective ECU i. R_i is used to derive the pseudo-random offsets, denoted $\hat{R}_i^j, j \in \mathbf{SL}_i$, for polynomial constructions. This rules out the possibility for an insider ECU to deduce any co-member's secret information from the polynomials. As a result, we do not rely on the large number factorization problem for protocol security as is imposed by **SKT-F1**. Within one session, the R_i received from KS will be used by ECU i to derive the same $\hat{R}_i^j$ as KS did. This reduces the overhead challenge raised in **SKT-F2**.

Second, to cope with vehicular ECU's modest processing power, we choose to deliver a 128-bit secret key sk into 16 bytes. For each byte of sk, denoted $sk_{[b]}$ for $b \in [16]$, a separate polynomial $f_b(x)$ is generated so that $f_b(0) = sk_{[b]}$. As for the arithmetic operations, all byte-wise operations are performed in the $GF(2^8)$ field[1]. To ensure the feasibility of byte-wise arithmetic in $GF(2^8)$, we assume the group size t_j of any group $\mathbf{G}_j$ (i.e., ECUs subscribing message j) is below 128. While we stick to this constraint in this work, larger group sizes (and operation in larger finite fields) can be supported if ECUs are equipped with more capable processors.

Third, we further configure that KS and all ECUs pre-share N' auxiliary 128-bit vectors $\hat{x}_1, ..., \hat{x}_{N'}$ with $N' := \max(t_1, ..., t_M)$ and the following property: $\forall b \in [16]$, the b-th byte of $\hat{x}_1, ..., \hat{x}_N$ and $x_1, ..., x_{N'}$, denoted $\hat{x}_{1[b]}, ..., \hat{x}_{N'[b]}$ and $x_{1[b]}, ..., x_{N[b]} \in \{0,1\}^8$, differ from each other. $\hat{x}_1, ..., \hat{x}_{N'}$ will be used on a per-byte basis as the x-coordinates of the auxiliary polynomial points. This design choice improves communication efficiency at a manageable storage cost, as only the y-coordinates of the auxiliary polynomial points need to be delivered by the protocol.

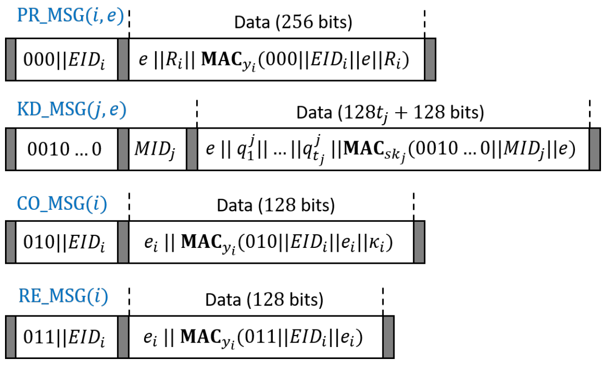
Finally, to fit into CAN/CAN-FD's message formats, we define four special protocol messages: *preparation* PR_MSG, *key delivery* KD_MSG, *confirmation* CO_MSG, and *request message* RE_MSG, with formats specified in Figure 5.

5.3 Protocol Workflow

For a new session, the SSKT protocol proceeds as follows:

- **Phase 1: Key Generation** Same as Phase 1 of SKDC.
- **Phase 2: Preparation** For each ECU i in **REL**, KS generates a random 128-bit vector R_i and sends it out in PR_MSG(i, e) which includes a y_i-keyed MAC. R_i is for initializing the *offset vector* for ECU i, denoted $\hat{R}_i$.
  When ECU i receives PR_MSG(i, e), it obtains the epoch e and verifies $e > e_i$. It then validates the received R_i with the MAC. If validation passes, it initializes its own $\hat{R}_i$ as R_i and updates e_i to e.
- **Phase 3-1: Key Delivery** For each message j in **RML**, KS generates the payload of KD_MSG(j, e) as follows. First, for each ECU $i \in G_j$, KS updates the offset vector for message j: $\hat{R}_i \leftarrow \mathbf{Enc}_{y_i}(\hat{R}_i)$ where the encryption fulfills a keyed pseudo-random function. The new $\hat{R}_i$ is used as the offset vector of ECU i for the current message j. That is, $\hat{R}_i^j \leftarrow \hat{R}_i$. Second, $\forall b \in [16]$, KS generates a t_j-degree polynomial $f_b^j(x)$ with the following $t_j + 1$ points: $\{(x_{i[b]}, y_{i[b]} \oplus \hat{R}_{i[b]}^j)\}_{i \in G_j}$ and $(0, sk_{j[b]})$. Third, $\forall l \in [t_j]$, KS computes a 128-bit q_l^j that $q_{l[b]}^j = f_b^j(\hat{x}_{l[b]}), b \in [16]$. Note that $\{(\hat{x}_{l[b]}, q_{l[b]}^j)\}_{l \in [t_j]}$ will serve as the t_j auxiliary points for the recovery of $f_b^j(0)$ at the ECU end. Finally, KS sends out KD_MSG(j, e). The payload contains $\{q_l^j\}_{\in [t_j]}$, the epoch e, and a sk_j-keyed MAC of the arbitration field and e.
- **Phase 3-2: Key Recovery** When ECU i receives message KD_MSG(j, e) from the bus and $j \in \mathbf{SL}_i$, it obtains the epoch e and $\{q_l^j\}_{\in [t_j]}$ and validates $e = e_i$. Then $\forall b \in [16]$, it computes $f_b^j(0)$ by interpolating on the following $t_j + 1$ points: the public auxiliary points $\{(\hat{x}_{l[b]}, q_{l[b]}^j)\}_{l \in [t_j]}$ and the secret

[1]Computation in a prime field with normal modular arithmetic (eg., $GF(251)$ for byte operations) is arguably faster than computation in $GF(2^8)$ for memory-constrained processors that do not support full multiplication and division tables. However, prime fields like $GF(251)$ do not make full use of a byte and thus have reduced security.



PR_MSG(i, e)
Data (256 bits)
$000||EID_i$
$e\,||R_i||\,\mathbf{MAC}_{y_i}(000||EID_i||e||R_i)$
KD_MSG(j, e)
Data ($128t_j + 128$ bits)
$0010 \dots 0$
MID_j
$e\,||\,q_1^j||\dots||q_{t_j}^j\,||\mathbf{MAC}_{sk_j}(0010\dots0||MID_j||e)$
CO_MSG(i)
Data (128 bits)
$010||EID_i$
$e_i\,||\,\mathbf{MAC}_{y_i}(010||EID_i||e_i||\kappa_i)$
RE_MSG(i)
Data (128 bits)
$011||EID_i$
$e_i\,||\,\mathbf{MAC}_{y_i}(011||EID_i||e_i)$

Figure 5: Protocol message formats of SSKT.

point $(x_{i[b]}, y_{i[b]} \oplus \hat{R}^j_{i[b]})$, in which $\hat{R}^j_i$ can be derived locally from R_i in the same way as KS did it. For convenience, we abbreviate the t_j+1 points as $(u_1, v_1), (u_2, v_2), \cdots, (u_{t_j+1}, v_{t_j+1})$ and the interpolation proceeds as follows:

$$f^j_b(0) = \sum_{m\in[t_j+1]} v_m \left(\prod_{n\in[t_j+1]/\{m\}} \frac{u_n}{u_n - u_m} \right) \quad (1)$$

The product term on the right is known as Lagrange coefficient. All arithmetic operations are performed in $GF(2^8)$. I.e., summation and subtraction are XORs; multiplication is done modulo an irreducible polynomial. The session key sk_j is obtained by concatenating $f^j_b(0)$ for all $b \in [16]$. ECU i then validates the recovered sk_j with the MAC: If pass, it accepts sk_j as the session key for MID_j; If fail, it sends out RE_MSG(i) to request KS to resend the KD_MSGs.

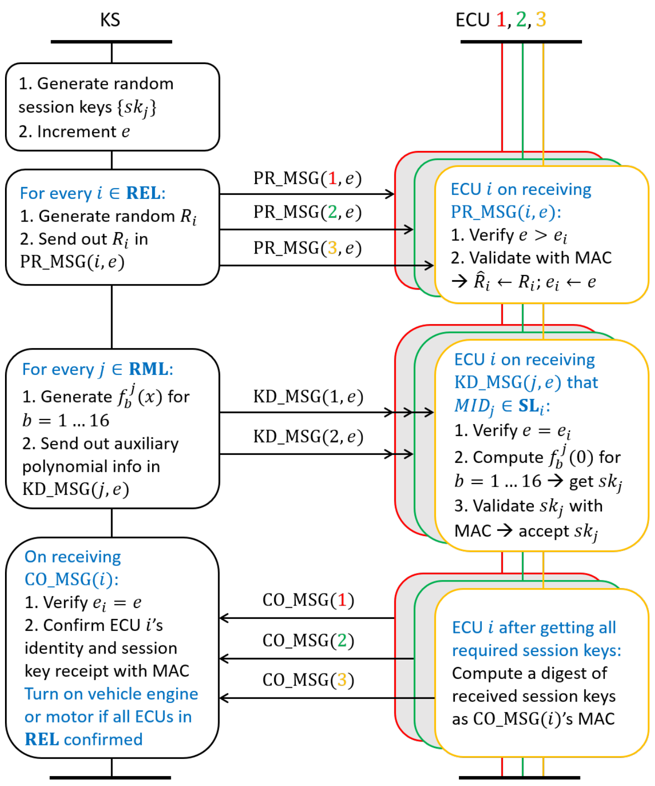
- **Phase 4: Confirmation** Same as Phase 3 of SKDC.

An execution example of SSKT is illustrated in Figure 6, in which we assume all PR_MSGs pass verification in the first try. For practical deployment in CAN bus, all three protocol messages need to break out into subsequent messages. Specially, the KD_MSGs have variable payload sizes and need to break out also in CAN-FD if the payload size surpasses 512 bits. To reduce computation workload at a ECU of modest computing power, the finite field arithmetic can be implemented with lookup tables and the key recovery phase can be realized efficiently with pre-computed Lagrange coefficients, which we will elaborate in §6.

For on-demand ECUs (those not checked in **REL** but switch on during normal operation), we let them obtain the session keys through the request-delivery mechanism of SKDC's. That is, after on-demand ECU k sends out a RE_MSG, KS sends back KD_MSGs in the SKDC format encrypted with x_k, instead of resending the KD_MSG of SSKT's. This is because SSKT's KD_MSG is designed to take advantage of the broadcast nature of CAN/CAN-FD bus for communication efficiency; SKDC's KD_MSG has shorter data payload length and is more efficient for one-on-one session key deliveries.

5.4 Security Analysis

We analyze the security of SSKT from the same three aspects as was done for SKDC.



KS
ECU 1, 2, 3
1. Generate random session keys $\{sk_j\}$
2. Increment e
For every $i \in$ **REL**:
1. Generate random R_i
2. Send out R_i in PR_MSG(i, e)
PR_MSG(1, e)
PR_MSG(2, e)
PR_MSG(3, e)
ECU i on receiving PR_MSG(i, e):
1. Verify $e > e_i$
2. Validate with MAC → $\hat{R}_i \leftarrow R_i$; $e_i \leftarrow e$
For every $j \in$ **RML**:
1. Generate $f^j_b(x)$ for $b = 1 \dots 16$
2. Send out auxiliary polynomial info in KD_MSG(j, e)
KD_MSG(1, e)
KD_MSG(2, e)
ECU i on receiving KD_MSG(j, e) that $MID_j \in$ **SL**$_i$:
1. Verify $e = e_i$
2. Compute $f^j_b(0)$ for $b = 1 \dots 16$ → get sk_j
3. Validate sk_j with MAC → accept sk_j
On receiving CO_MSG(i):
1. Verify $e_i = e$
2. Confirm ECU i's identity and session key receipt with MAC
Turn on vehicle engine or motor if all ECUs in **REL** confirmed
CO_MSG(1)
CO_MSG(2)
CO_MSG(3)
ECU i after getting all required session keys:
Compute a digest of received session keys as CO_MSG(i)'s MAC

Figure 6: Example workflow of SSKT. Message subscription follows Fig. 2.

First, session key correctness is equivalent to the integrity and authenticity of KD_MSG. A KD_MSG(j, e) contains the auxiliary coordinates and a 64-bit hash computed with the session key sk_j. Once ECU i recovers sk_j, its authenticity can be verified with the attached MAC. An attacker who wishes to forge a valid KD_MSG(j, e) needs to coincide a MAC before KS broadcasts the authentic one, which yields expected 2^{63} MAC evaluations to succeed. Replay attack resistance is achieved via the epoch mechanism, similar to that in SKDC.

Second, recovering the session key from KD_MSG(j, e) is equivalent to reconstructing the t_j-degree polynomial $f^j_b(x), \forall b \in [16]$. To do so, an ECU needs to interpolate on t_j auxiliary points and its secret point. An attacker with arbitrary computing capability gains zero information about $f^j_b(x)$ knowing only the auxiliary points. This is commonly referred to as information-theoretical security. Nonetheless, since session key sk_j is the only secret input to the MAC of KD_MSG(j, e), the attacker can brute-force the session key until a valid MAC is found. After excluding the t_j auxiliary y-coordinates for each byte, the attacker expects $\frac{1}{2}(2^8 - t_j)^{16} \in (2^{111}, 2^{127})$ MAC evaluations to succeed, which retains sufficient security for a single session. The result comes from the fact that $t_j \leq N'$, and N' is the number of auxiliary vectors and assumed to be smaller than 128. For the ECU key pair (x_i, y_i), the confidentiality of y_i can be analyzed with respect to the MAC of PR_MSG, CO_MSG and RE_MSG similar to that of SKDC. The confidentiality of x_i reduces to that of y_i (to insider). That is, the information-theoretical security of polynomial secret recovery means no information about x_i can be inferred from the auxiliary points. For a malicious insider knowing session key sk_j and the polynomial $f_b^j(x), \forall b \in [16]$, he still cannot locate ECU i's secret point on the polynomial without knowing y_i. As a result, a brute-force attacker expects 2^{127} MAC evaluations to find y_i or $\frac{1}{2}(2^8 - N')^{16} \in (2^{111}, 2^{127})$ guesses to coincide with x_i.

Lastly, analysis on ECU entity authentication and liveness follows from SKDC's since SSKT adopts the same confirmation mechanism.

6 IMPLEMENTATION

We implemented the SKDC and SSKT protocols for CAN bus deployment[2] with Arduino IDE [35] and the CAN Bus Shield library by Seeed Studio [36]. For each protocol, the implementation consists of two C++ programs: *keyserver* and *ecu*. Execution of the *ecu* program is driven by the receipt of protocol messages from KS who runs the *keyserver* program. The following design choices are made with the aim of maximizing system efficiency:

1) Using Lightweight Cryptography We use the Arduino Cryptography Library (ACL) [41] for standard cryptographic functions. ACL provides three versions of AES128 cipher: normal (AES128), small (AESSmall128), and tiny (AESTiny128), with main differences in cipher state size (RAM usage) and key setup time. Given that vehicular ECU can be constrained in memory and computing power, we choose AESSmall128 for en/decryption of session keys in SKDC for reduced RAM usage and relatively fast cipher speed. To partially compensate for the memory cost of storing precomputed Langrange coefficients (to introduce shortly), we choose AESTiny128 for the generation of random offset (R_i^j) in SSKT for minimal RAM usage in the *ecu* program. All en/decryption are in ECB mode. For MAC function in both protocols, we opt for the keyed-mode BLAKE2s hash. BLAKE2s is the 256-bit variant of BLAKE2 [1], a lightweight hash function specifically designed for speed in software and RAM saving while attaining SHA-3 level security[3].

2) Using Lookup Tables for $GF(2^8)$ Arithmetic Comparing to the standard AES en/decryption functions used by SKDC, SSKT relies on $GF(2^8)$ arithmetic for generating secret shares and polynomial recovery shown in Eq. (1), which can be optimized for computation efficiency using lookup tables. Our implementation does not use full multiplication or division tables in $GF(2^8)$ due to the presumed memory constraint of ECU (256-by-256 bytes for each table). Instead, we make use of intermediate lookup tables of multiplicative inverse, exponentiation, and logarithm (16-by-16 bytes for each table). Multiplication is efficiently implemented with exponentiation and logarithm on generator 0x03. Division is implemented with multiplication and multiplicative inverse.

3) Precomputing Lagrange Coefficients The Lagrange coefficients in Eq. (1) can be pre-computed to speed up key recovery during the SSKT runtime of *ecu* program, with computation complexity linear to N. This is because the auxiliary vectors $\hat{x}_1, ..., \hat{x}_{N'}$, used on per-byte basis as x-coordinates of auxiliary polynomial points, are pre-determined and known to all. In comparison, computing Eq. (1) from scratch would incur complexity quadratic to N. The memory allocation at each ECU for the pre-computed Lagrange coefficients is $16(N + 1)$ bytes.

4) Timing of Protocol Execution In the *keyserver* program of both SKDC and SSKT, if multiple session keys are to be delivered to one ECU in the key delivery phase, a wait period denoted *KdDelay* is enforced after sending each KD_MSG. For SSKT, an additional wait period denoted *PrDelay* is enforced after sending the last PR_MSG in the preparation phase. The reason behind the waiting mechanism is as follows. Since KS has much higher processing power than a normal ECU, the latter takes longer to process a protocol message (eg., MAC validation and key recovery for KD_MSG) than KS generating that message. In our Arduino-based implementation, ECU reads from the CAN Rx buffer only after the incumbent message gets finished processing. If KS sends messages too frequently, the ECU will miss out some of them due to its limited Rx buffer size (we do not implement any queuing mechanism for CAN messages in the *ecu* program for memory saving). In response, the waiting mechanism described above ensures that the ECU receives all required protocol messages in the correct order and format. For achieving the best time efficiency, the minimum possible values of *KdDelay* and *PrDelay* can be determined by experimenting with the actual CAN bus system and used for evaluation, as we will show in §7.1.

7 EVALUATION

The evaluation contains two parts. In §7.1 we construct a CAN bus test platform and evaluate protocol runtimes for an entire key distribution session. In §7.2 we benchmark the two protocols with respect to protocol functionalities and perform large-scale extrapolation analyses on their performance in CAN/CAN-FD buses.

7.1 Runtime Evaluation with Test Platform

We set up a CAN bus test platform with an Arduino Due board (32-bit processor, 84MHz clock) as KS and six Arduino Uno broads (8-bit processor, 16MHz clock) as ECU nodes. A photo of the test platform is shown in Figure 7. A Seeed Studio CAN bus shield was attached to each board to fulfill the link and physical layer functions of CAN. All shields were connected to a common bus from the CAN_L, CAN_H interface. A 120Ω terminal resistor was added to each end of the bus. We note that the test platform was for CAN bus only since CAN-FD-compatible Arduino accessories were not commercially available at the time of writing.

For the evaluation of each protocol, we used protocol runtime, measured from the generation of session keys to the receipt of all CO_MSGs by KS, as the performance metric. The CAN bit rate was fixed at 500kb/s. The *keyserver* program was configured to distribute session keys for M message IDs to N ECUs in one session, with every message ID being subscribed by all N ECUs. The *keyserver* and *ecu* programs were uploaded to KS and each ECU node respectively.

The protocol runtimes of SKDC and SSKT for one session under different M, N are shown in Figure 8. In all cases, we find the minimum possible *PrDelay* for SSKT is 5.7 ms and it is insensitive to M or N. When $M = 1$, *KdDelay* is not needed for both protocols. SKDC achieves better result than SSKT because of the overhead brought by latter's preparation phase. When $M = 6$, the minimum possible *KdDelay* for SKDC is 6.8 ms for any N while for SSKT

[2]Source code and documentation: https://github.com/yang-sec/CAN-SessionKey.
[3]BLAKE, the original version of BLAKE2, was a finalist in the SHA-3 competition.

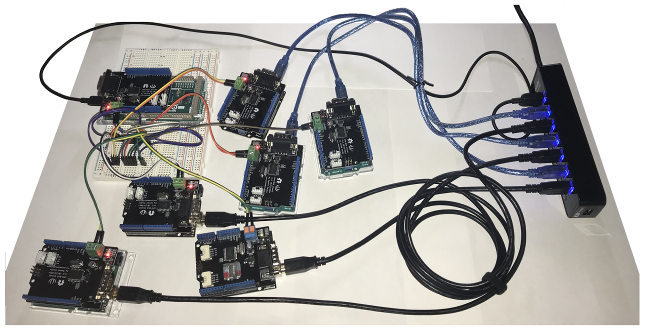


Figure 7: CAN bus test platform setup.

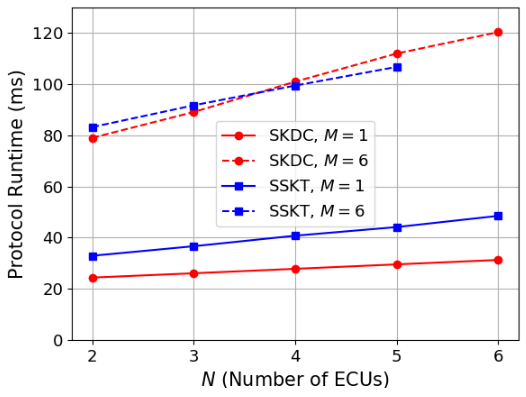


Figure 8: Protocol runtime with CAN bus test platform.

it is $\{6.5, 6.2, 5.5, 4.4\}$ms for $N = \{2, 3, 4, 5\}$ respectively. Also the gap between SKDC and SSKT diminishes for $M = 6$ and the latter starts to show advantage after $N = 4$. It is anticipated that SSKT's advantage will scale up for larger M and N, contributed by its better communication efficiency in the key delivery phase and the amortization effect of its one-time preparation overhead. With our test platform, however, the scale for SSKT experiment stops at $M = 6, N = 5$ due to the memory constraint of Arduino Uno.

7.2 Extrapolation Analyses

In order to bypass the hardware limitation of the test platform, we perform benchmark tests on protocol functionalities and use them to extrapolate protocol performance in large-scale CAN/CAN-FD bus networks with respect to ECU computation workload and communication overhead.

ECU Computation Workload The runtimes for single cryptographic computations in Arduino Uno and Arduino Due (both simulating ECU not KS) are shown in Table 1. It is observed that one-time hash operations (keyed reset and finalize) are much more costly in Uno than in Due. We speculate the reason is that the modest 8-bit processor of Uno does not support the same level of parallelization in these hash operations than the 32-bit processor of Due. Polynomial recovery computations, which are boosted by the usage of $GF(2^8)$ lookup tables and pre-computed Lagrange coefficients, are significantly more efficient than AES en/decryption and become too minimal to be measured in Due.

Table 1: Cryptographic computation runtimes (μs) on Arduino Uno and Arduino Due (simulating ECU not KS)

Operation	Uno	Due
AESSmall128-ECB Set Key	131.64	23.66
AESSmall128-ECB Encrypt (per byte)	42.40	7.06
AESSmall128-ECB Decrypt (per byte)	73.66	12.33
AESTiny128-ECB Set Key	9.98	1.25
AESTiny128-ECB Encrypt (per byte)	42.39	7.23
BLAKE2s Keyed Reset	3512.94	55.09
BLAKE2s Hash (per byte)	54.61	0.80
BLAKE2s Finalize	3508.25	53.14
Degree-2 Polynomial $f(0)$ Recovery (per byte)	10.40	~ 0
Degree-5 Polynomial $f(0)$ Recovery (per byte)	19.86	~ 0
Degree-10 Polynomial $f(0)$ Recovery (per byte)	33.56	~ 0

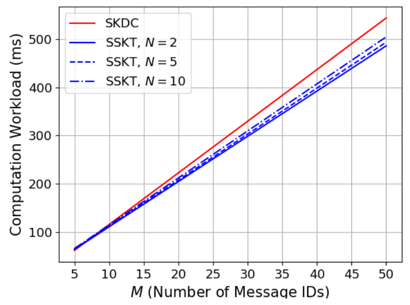
Table 2: Communication overheads of SKDC and SSKT protocol messages

		Message size (In CAN bits)	Message count
SKDC (CAN):	KD_MSG	524	MN
	CO_MSG	222	N
SKDC (CAN-FD):	KD_MSG	105	MN
	CO_MSG	60	N
SSKT (CAN):	PR_MSG	444	N
	KD_MSG	$262(1+N)$	M
	CO_MSG	222	N
SSKT (CAN-FD):	PR_MSG	86	N
	KD_MSG	avg. $60 + 39N$	M
	CO_MSG	60	N

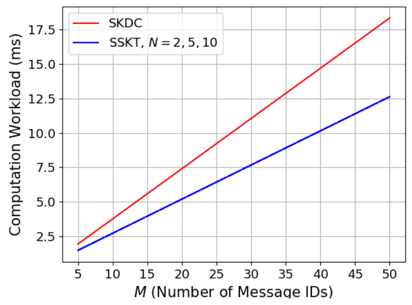
The overall ECU computation workloads of a complete session for different M, N are extrapolated from the cryptographic computation runtimes. The results of Uno experiment and Due experiment are shown in Figure 9(a) and 9(b) respectively. We observe that in the Uno experiment the ECU workload of SSKT slightly increases in N but generally stays below that of SKDC when $M \geq 10$. For the Due experiment (representing a more powerful ECU), SSKT achieves a bigger reduction in ECU workload than SKDC does.

Communication Overhead The size and count of each protocol message are shown in Table 2. Message count represents the occurrence of the protocol message in one session. Message size represents the communication overhead in CAN bits brought by one such protocol message and is calculated based on follows:

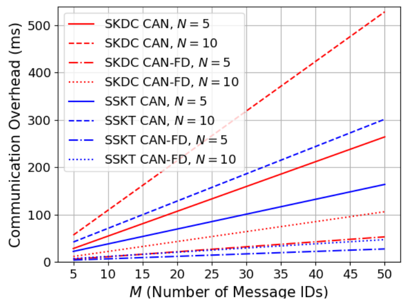
- CAN and CAN-FD message fields excluding the data fields and CAN-FD's CRC field have fixed sizes. The size of a CAN-FD CRC field depends on the preceding data field size [32]. The data field sizes of follow Figure 3 and 5.
- We assume the DLC, data and CRC fields of a CAN-FD frame are transmitted at 5 times of the CAN bit rate, which is inline with the standard specification [18]. All other fields



(a) ECU computation workload (Arduino Uno)



(b) ECU computation workload (Arduino Due)



(c) Communication overhead

Figure 9: Extrapolation analyses.

are transmitted at the CAN bit rate. The message size results are denominated in CAN bits.
- If a protocol message needs to be broken into separate CAN or CAN-FD frames for transmission, the message size result accounts for all the break-out frames.

With the above criteria and the CAN bit rate set at 500kb/s, the overall communication overheads of a complete session for different M, N are extrapolated and shown in Figure 9(c). We observe that SSKT yields a significantly reduced overall communication overhead than SKDC for any bus type and N, when $M \geq 10$. For example, given $N = 10$, $M = 50$ and CAN-FD bus, which we speculate would be an appropriate automotive network scenario, the communication overheads of SKDC and SSKT are 106 ms and 47 ms respectively, marking a 56% advantage for SSKT.

8 DISCUSSION

8.1 Performance Bottlenecks

Compared to computation workload, communication overhead is less flexible and has limited room for improvement, since it is largely determined by the underlying CAN/CAN-FD protocol and physical-layer specification. If ECU processing capability continues to improve (eg., using Arduino Due instead of Uno in our case) and the automotive industry imposes more stringent timing requirements on key establishment, the inflexible communication overhead will become a major performance bottleneck. To this end, we consider SSKT's advantage in communication efficiency a highlight for its adoption.

8.2 Memory Cost

SSKT's superior computation efficiency comes at the cost of extra memory usage. SSKT needs to store the pre-distributed N' auxiliary vectors of x-coordinates, requiring $16N'$ bytes of RAM. In our implementation, SSKT also needs to store the $GF(2^8)$ lookup tables and pre-compute Lagrange coefficients to speed up polynomial recovery, costing another $(768 + 16(N + 1))$ bytes of RAM. Although nowadays memory has become fairly affordable even for embedded devices, the trade-off between computation efficiency (time) and memory usage (space) itself is an interesting topic and deserves more attention from the resource optimization perspective.

8.3 Deficiency and Future Work

The current design of SSKT supports 128 or fewer ECUs in one messaging group, due to the polynomial computation within $GF(2^8)$. Combining with the consideration on memory cost, a proper deployment scenario for our current SSKT implementation is a CAN/CAN-FD bus network of controlled ECU population for each message group. Although a modern passenger car typically has no more than 80 ECUs [9], of which those connected by a CAN/CAN-FD bus are even fewer, we still consider scalability in ECU quantity an important onward issue. For SSKT to support larger ECU messaging groups, we can dissect the session key into fewer and longer blocks (eg., 2-Bytes). Then polynomial computation shall be performed in a larger finite field (eg., $GF(2^{16})$) which supports much bigger group sizes (eg., up to 2^{15}). This in turn requires higher processing capability from ECUs and more efficient implementations of finite field arithmetic, as lookup tables could be too large to store.

Lastly, our hardware evaluation is not able to capture the impact of tamper-resistant memory usage, which would need ECU implementation with trusted platform modules (TPM). We will address the above challenges in future work.

9 RELATED WORK

9.1 Legacy CAN/CAN-FD Message Authentication & Key Management Schemes

Recognizing the lack of built-in authentication mechanisms of automotive communication networks and its security implication, various message authentication schemes have been proposed prior to AUTOSAR specification with a focus on CAN bus. [15, 26, 28] achieve point-to-point authentication wherein each pair of sender and receiver ECUs share a unique MAC key. However, point-to-point authentication schemes are impractical for the resource-constrained in-vehicle ECUs for two reasons. First, they need significant resource at each ECU to manage pairwise keys. Second, each broadcast message needs to carry multiple MACs intended for different recipients, resulting in communication inefficiency.

In contrast, [11, 14, 21, 33, 39, 40, 43] aim for group authentication, i.e., each group of ECUs share a unique MAC key for internal message authentication. These schemes differ in the grouping method of ECUs. [21, 40] allows for a single group of ECUs for MAC purposes. [14, 33, 43] group ECUs based on designated security levels, each level is assigned a MAC key. [11] assumes arbitrary ECU groupings that is assigned a MAC key with a help of a key server. [39] separates message IDs into security groups that in each group one ECU is designated as the sole sender for distributing keying materials to all members.

Noticeably, these legacy schemes commonly adopt arbitrarily MAC truncation (some are 8 or 16 bits) to cope with the limited payload capacity. Nonetheless, these works provide valuable lessons on deploying security mechanisms in CAN/CAN-FD which are often echoed in later AUTOSAR-compliant designs [27, 30, 38].

9.2 Secret-sharing-based Group Key Establishment

Secret sharing has been used for key establishment in previous work. In early works [5, 6, 10], a key generation paradigm is used in that group keys can be derived from the pre-distributed secret shares. To reduce storage for pre-distributed information and distinguish sessions, later works [3, 4, 12, 22, 23] let the key server to actively generate a new secret key and deliver the secret shares to the group members, although they differ in the specific share delivery and secret reconstruction mechanisms. The SKT primitive adopted in our paper follows the second paradigm and further supports variable group size and the key authentication, which fits our automotive setting.

10 CONCLUSION

In this paper we address the practical issue of session key establishment for message authentication purposes in automotive communication networks. Under a proposed AUTOSAR-compliant key management architecture, we construct two protocols: the baseline SKDC and the communication-efficient SSKT, both of which are customized for deployment in CAN and CAN-FD buses. We implemented the proposed SKDC and SSKT protocols on commercial microcontroller boards and evaluated their performance with hardware experiment and extrapolation analysis. The result shows that SSKT achieves better computation and communication efficiency at scale which results in a lower overall protocol runtime, at the cost of increased ECU memory footprint. In future work we will continue to improve the proposed protocols with a special focus on memory usage control and scalability in network size. In the evolving landscape of security challenges in modern vehicles, we hope the proposed architecture design, key distribution protocols, and discussion on practical issues will provide useful insights towards deployable security for vehicular systems.

ACKNOWLEDGMENTS

This work was supported in part by US National Science Foundation under grant CNS-1837519 and by Virginia Commonwealth Cyber Initiative (CCI).

REFERENCES

[1] Jean-Philippe Aumasson, Samuel Neves, Zooko Wilcox-O'Hearn, and Christian Winnerlein. 2013. BLAKE2: simpler, smaller, fast as MD5. In *International Conference on Applied Cryptography and Network Security*. Springer, Berlin, Heidelberg, 119–135.

[2] AUTOSAR. 2017. AUTOSAR Release 4.2.2: Specification of Module Secure Onboard Communication. https://www.autosar.org/fileadmin/user_upload/standards/classic/4-3/AUTOSAR_SWS_SecureOnboardCommunication.pdf

[3] Amos Beimel et al. 1996. *Secure schemes for secret sharing and key distribution.* Technion-Israel Institute of technology, Haifa, Israel.

[4] Shimshon Berkovits. 1991. How to broadcast a secret. In *Workshop on the Theory and Application of of Cryptographic Techniques*. Springer, Berlin, Heidelberg, 535–541.

[5] Rolf Blom. 1984. An optimal class of symmetric key generation systems. In *Workshop on the Theory and Application of of Cryptographic Techniques*. Springer, Berlin, Heidelberg, 335–338.

[6] Carlo Blundo, Alfredo De Santis, Amir Herzberg, Shay Kutten, Ugo Vaccaro, and Moti Yung. 1992. Perfectly-secure key distribution for dynamic conferences. In *Annual international cryptology conference*. Springer, Berlin, Heidelberg, 471–486.

[7] LIN Consortium. 2010. LIN Specification Package, Revision 2.2A.

[8] MOST Cooperation. 2004. MOST Specification Revision 2.3.

[9] Christof Ebert and Capers Jones. 2009. Embedded software: Facts, figures, and future. *Computer* 42, 4 (2009), 42–52.

[10] Amos Fiat and Moni Naor. 1993. Broadcast encryption. In *Annual International Cryptology Conference*. Springer, Berlin, Heidelberg, 480–491.

[11] Bogdan Groza, Stefan Murvay, Anthony Van Herrewege, and Ingrid Verbauwhede. 2012. LiBrA-CAN: a lightweight broadcast authentication protocol for controller area networks. In *International Conference on Cryptology and Network Security*. Springer, Berlin, Heidelberg, 185–200.

[12] Lein Harn. 1995. Efficient sharing (broadcasting) of multiple secrets. *IEE Proceedings-Computers and Digital Techniques* 142, 3 (1995), 237–240.

[13] L. Harn and C. Lin. 2010. Authenticated Group Key Transfer Protocol Based on Secret Sharing. *IEEE Trans. Comput.* 59, 6 (2010), 842–846.

[14] Oliver Hartkopp, Cornel Reuber, and Roland Schilling. 2012. Message authenticated CAN. In *10th Int. Conf. on Embedded Security in Cars (ESCAR 2012), Berlin, Germany*.

[15] Ahmed Hazem and HA Fahmy. 2012. LCAP - a lightweight can authentication protocol for securing in-vehicle networks. In *10th Int. Conf. on Embedded Security in Cars (ESCAR 2012), Berlin, Germany*, Vol. 6.

[16] ISO. 2006. *ISO 11898-3:2006 - Road vehicles - Controller area network (CAN) - Part 3: Low-speed, fault-tolerant, medium-dependent interface.* Standard. International Organization for Standardization, Geneva, Switzerland.

[17] ISO. 2015. *ISO 11898-1:2015 - Road vehicles - Controller area network (CAN) - Part 1: Data link layer and physical signalling.* Standard. International Organization for Standardization, Geneva, Switzerland.

[18] ISO. 2016. *ISO 11898-2:2016 - Road vehicles - Controller area network (CAN) - Part 2: High-speed medium access unit.* Standard. International Organization for Standardization, Geneva, Switzerland.

[19] Karl Koscher, Alexei Czeskis, Franziska Roesner, Shwetak Patel, Tadayoshi Kohno, Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, et al. 2010. Experimental security analysis of a modern automobile. In *2010 IEEE Symposium on Security and Privacy*. IEEE, New York, NY, USA, 447–462.

[20] Sekar Kulandaivel, Tushar Goyal, Arnav Kumar Agrawal, and Vyas Sekar. 2019. CANvas: fast and inexpensive automotive network mapping. In *28th USENIX Security Symposium (USENIX Security 19)*. USENIX Association, Berkeley, CA, 389–405.

[21] Ryo Kurachi, Yutaka Matsubara, Hiroaki Takada, Naoki Adachi, Yukihiro Miyashita, and Satoshi Horihata. 2014. CaCAN-centralized authentication system in CAN (controller area network). In *14th Int. Conf. on Embedded Security in Cars (ESCAR 2014), Hamburg, Germany*.

[22] Chi Sung Laih, Jau Yien Lee, and Lein Harn. 1989. A new threshold scheme and its application in designing the conference key distribution cryptosystem. *Inform. Process. Lett.* 32, 3 (1989), 95–99.

[23] Chih-Hung Li and Josef Pieprzyk. 1999. Conference key agreement from secret sharing. In *Australasian Conference on Information Security and Privacy*. Springer, Berlin, Heidelberg, 64–76.

[24] Rainer Makowitz and Christopher Temple. 2006. Flexray-a communication network for automotive control systems. In *2006 IEEE International Workshop on Factory Communication Systems*. IEEE, New York, NY, USA, 207–212.

[25] Charlie Miller and Chris Valasek. 2015. Remote exploitation of an unaltered passenger vehicle. *Black Hat USA* 2015 (2015), 91.

[26] Dennis K Nilsson, Ulf E Larson, and Erland Jonsson. 2008. Efficient in-vehicle delayed data authentication based on compound message authentication codes. In *2008 IEEE 68th Vehicular Technology Conference*. IEEE, New York, NY, USA, 1–5.

[27] Stefan Nürnberger and Christian Rossow. 2016. –vatiCAN– vetted, authenticated CAN bus. In *International Conference on Cryptographic Hardware and Embedded Systems*. Springer, Berlin, Heidelberg, 106–124.

[28] Hisashi Oguma, Akira Yoshioka, Makoto Nishikawa, Rie Shigetomi, Akira Otsuka, and Hideki Imai. 2008. New attestation based security architecture for in-vehicle communication. In *IEEE GLOBECOM 2008-2008 IEEE Global Telecommunications Conference*. IEEE, New York, NY, USA, 1–6.

[29] Mert D Pesé, Troy Stacer, C Andrés Campos, Eric Newberry, Dongyao Chen, and Kang G Shin. 2019. LibreCAN: Automated CAN Message Translator. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. ACM, New York, NY, USA, 2283–2300.

[30] Andreea-Ina Radu and Flavio D Garcia. 2016. LeiA: A lightweight authentication protocol for CAN. In *European Symposium on Research in Computer Security (ESORICS 2016)*. Springer, Cham, 283–300.

[31] Robert Bosch GmbH. 1991. CAN Specification Version 2.0.

[32] Robert Bosch GmbH. 2012. CAN with Flexible Data-Rate Version 1.0.

[33] Hendrik Schweppe, Yves Roudier, Benjamin Weyl, Ludovic Apvrille, and Dirk Scheuermann. 2011. Car2x communication: securing the last meter-a cost-effective approach for ensuring trust in car2x applications using in-vehicle symmetric cryptography. In *2011 IEEE Vehicular Technology Conference (VTC Fall)*. IEEE, New York, NY, USA, 1–5.

[34] Adi Shamir. 1979. How to share a secret. *Commun. ACM* 22, 11 (1979), 612–613.

[35] Arduino Software. 2020. Arduino IDE Documentation. https://www.arduino.cc/en/Guide

[36] Seeed Studio. 2018. CAN BUS Shield. https://github.com/Seeed-Studio/CAN_BUS_Shield

[37] Shane Tuohy, Martin Glavin, Ciarán Hughes, Edward Jones, Mohan Trivedi, and Liam Kilmartin. 2014. Intra-vehicle networks: A review. *IEEE Transactions on Intelligent Transportation Systems* 16, 2 (2014), 534–545.

[38] Jo Van Bulck, Jan Tobias Mühlberg, and Frank Piessens. 2017. VulCAN: Efficient component authentication and software isolation for automotive control networks. In *Proceedings of the 33rd Annual Computer Security Applications Conference*. ACM, New York, NY, USA, 225–237.

[39] Anthony Van Herrewege, Dave Singelee, and Ingrid Verbauwhede. 2011. CANAuth - a simple, backward compatible broadcast authentication protocol for CAN bus. In *ECRYPT Workshop on Lightweight Cryptography*, Vol. 2011.

[40] Qiyan Wang and Sanjay Sawhney. 2014. VeCure: A practical security framework to protect the CAN bus of vehicles. In *2014 International Conference on the Internet of Things (IOT)*. IEEE, New York, NY, USA, 13–18.

[41] Rhys Weatherley. 2018. Arduino Cryptography Library. https://rweather.github.io/arduinolibs/crypto.html

[42] Haohuang Wen, Qingchuan Zhao, Qi Alfred Chen, and Zhiqiang Lin. 2020. Automated Cross-Platform Reverse Engineering of CAN Bus Commands From Mobile Apps. In *The 2020 Network and Distributed System Security Symposium (NDSS'20), San Diego, CA, USA.*

[43] Samuel Woo, Hyo Jin Jo, In Seok Kim, and Dong Hoon Lee. 2016. A practical security architecture for in-vehicle CAN-FD. *IEEE Transactions on Intelligent Transportation Systems* 17, 8 (2016), 2248–2261.

[44] Werner Zimmermann and Ralf Schmidgall. 2006. *Bussysteme in der Fahrzeugtechnik*. Springer, Vieweg, Wiesbaden.