SPARK

Secure Privacy-Preserving Anonymous Swarm Attestation for In-Vehicle Networks

Hellemans, Wouter; El Kassem, Nada; Rabbani, Md Masoom; Dushku, Edlira; Chen, Liqun; Braeken, An; Preneel, Bart; Mentens, Nele

Published in:
10th IEEE European Symposium on Security and Privacy 2025 (EuroS&P 2025)

Publication date:
2025

Document Version
Accepted author manuscript, peer reviewed version

SPARK: Secure Privacy-Preserving Anonymous Swarm Attestation for In-Vehicle Networks

Wouter Hellemans*, Nada El Kassem†, Md Masoom Rabbani‡, Edlira Dushku§,
Liqun Chen†, An Braeken¶, Bart Preneel*, Nele Mentens*‖
*COSIC, ESAT, KU Leuven, Leuven, Belgium, {firstname.lastname}@kuleuven.be
†University of Surrey, Surrey, United Kingdom, {firstname.lastname}@surrey.ac.uk
‡Chalmers University of Technology & Univ. of Gothenburg, Gothenburg, Sweden, {firstname.lastname}@chalmers.se
§Aalborg University, Copenhagen, Denmark, edu@es.aau.dk
¶Vrije Universiteit Brussel, Belgium, {firstname.lastname}@vub.be
‖LIACS, Leiden University, Leiden, The Netherlands

***Abstract*—In recent years, vehicles have evolved into cyber-physical autonomous systems that rely on sensor data from various sources within the vehicle. With the emergence of Vehicle-to-Everything (V2X) technology, the scope of the collaborative functionality in vehicles is now expanding to the inter-vehicular level. To support these modern capabilities, the complexity of the Electronic Control Units (ECUs) and the In-Vehicle Network (IVN) architecture is rapidly increasing. As a result, IVNs are now swarms of devices that communicate safety-critical data. Unfortunately, current vehicular networks lack security, opening the path to numerous cyberattacks. A typical solution for verifying the integrity of multiple devices is swarm attestation. However, in a typical IVN setting, only the Original Equipment Manufacturer (OEM) has access to the legitimate configuration of the ECUs and does not want to disclose this information due to intellectual property and security concerns. Therefore, state-of-the-art swarm attestation schemes, which do not provide privacy guarantees, are unsuitable for IVNs.**

This paper proposes Secure Privacy Preserving Anonymous Swarm Attestation for In-Vehicle Networks (SPARK), which builds upon a novel group signature scheme to enable privacy-preserving, anonymous, and traceable swarm attestation of IVNs. We validate SPARK through a proof-of-concept implementation using a standardized hardware Trusted Platform Module (TPM 2.0) and representative hardware platforms. The results demonstrate the real-world applicability of SPARK.

***Index Terms*—Swarm attestation, In-vehicle networks, CAN, IoT security**

1. Introduction

With the ever-increasing demand for safety, automation, and user satisfaction, the number of collaborative functions in vehicles is rapidly growing. As such, modern vehicles are now becoming software-based cyber-physical systems driven by sensor data from various sources in the vehicle. In Advanced Driver-Assistance System (ADAS) scenarios, where vehicle sensor data like speed, distance, and camera data are shared for safety-critical decisions, maintaining the integrity of these sensors and gateways is crucial. If Electronic Control Units (ECUs) such as the Brake Control Module or Forward Collision Warning are compromised, they can transmit false data, potentially leading to collisions or system failures.

In recent years, it has been shown that vehicles are prone to cyberattacks [48], [59], [50], [51], [28]. One example that highlights the issue is the Jeep Cherokee hack [57], where researchers were able to inject arbitrary messages in the In-Vehicle Network (IVN) by reprogramming the firmware of the infotainment system, leading to a recall of 1.4 million vehicles. In this context, a central problem is effectively segmenting the network for security purposes. Indeed, collaborative functions such as ADAS pose a challenge in segmentation as they require data from multiple sources in the vehicle. Furthermore, the infotainment system - the largest attack surface in a modern vehicle [20] - by law has to display critical data to the user, compromising the intended security segmentation.

As IVNs continue to grow, new protocols have been proposed to cope with the increased bandwidth. Whereas the Controller Area Network (CAN) [17] has been the de-facto standard for three decades, recently, new generations of CAN, namely, CAN FD [18] and CAN XL [25], as well as other technologies (e.g., Automotive Ethernet (AE)) are finding adoption. Furthermore, to improve the efficiency of IVNs, a new network arrangement, known as zonal architecture [5], has been proposed. In this arrangement, the ECUs are grouped based on their physical location within the vehicle rather than their specific functions (as is the case in domain architectures). This approach reduces wiring costs and enables centralized processing through Zonal Gateways and a Central Gateway [5]. Additionally, this architecture opens up opportunities for emerging Vehicle-to-Everything (V2X) technology to facilitate collaborative functionalities between vehicles. However, this arrangement poses security challenges for IVNs as it typically does not provide physical segmentation of critical functions. Nevertheless, establishing trust between in-vehicle devices is crucial to enable collaborative functionalities between vehicles.

Modern vehicles can have over 150 ECUs and more than 100 million lines of code [72]; this increased complexity presents substantial security challenges. While state-of-the-art IVN security solutions (e.g., Network Intrusion Detection Systems) focus on detecting the anomalous behaviour of the connected devices, such approaches

typically only focus on the message communication and produce false positive/negative findings [52]. This necessitates security solutions that precisely verify the device integrity. One crucial method for ensuring firmware integrity is Remote Attestation (RA) [69]. Typically, a RA protocol involves an interactive process between a trusted party known as the Verifier ($\mathcal{V}$) and a potentially untrusted party referred to as the Prover (Prv). In this context, it is assumed that $\mathcal{V}$ and Prv have prior knowledge of each other, with $\mathcal{V}$ possessing accurate information about the Prv's legitimate configuration. During attestation, $\mathcal{V}$ sends a challenge ch, which prompts Prv to compute its firmware state (e.g., by computing the hash of the firmware) and relay the result back to $\mathcal{V}$. Various RA protocols have been proposed in literature, including *swarm attestation* approaches [6] that aim to attest collectively a large number of devices. However, such conventional RA approaches may not be applicable in vehicular settings with vehicles being attested by each other and roadside infrastructure. In the context of IVNs, the legitimate configuration of the ECUs is confidential and only known by the Original Equipment Manufacturer (OEM). Indeed, sharing information regarding the legitimate configuration of ECUs with third parties poses privacy challenges, security concerns, and intellectual property issues [55], [44]. Additionally, whenever any of the ECUs is deemed compromised, it is essential to identify the affected device for maintenance. To this end, modern vehicles require a beyond state-of-the-art RA mechanism that preserves individual ECU's privacy during attestation but also allows a trusted tracer authority to open the result in case of attestation failure.

Contribution. In this paper, we propose Secure Privacy Preserving Anonymous Swarm Attestation for In-Vehicle Networks (SPARK), a novel RA scheme that aims to provide efficient, privacy-preserving, anonymous, and traceable attestation for static swarm topologies, such as IVNs. SPARK enables trust in V2X settings by preserving the privacy of the individual ECUs while allowing public verification by any third-party, such as Road-Side Units (RSUs) or another vehicle. Specifically, in SPARK, we rely on the capabilities of Trusted Platform Module (TPM) 2.0 present in emerging vehicles [41] and a modified Direct Anonymous Attestation with Attributes (DAA-A) scheme [24]. SPARK considers the emerging zonal architecture in IVNs, in which a swarm of ECUs (IoT devices) is connected to Zonal Gateways (i.e., Edge devices) while grouped into branches. We summarize the main contributions as follows:

Evidence Privacy: In SPARK, each branch belonging to a swarm is able to provide verifiable evidence that guarantees the correctness of the swarm's operational state while not revealing any configuration of its connected devices to an external verifier. We achieve evidence privacy in a way that avoids using Zero-Knowledge Proofs (ZKPs) to hide the evidence, as this would increase the protocol's complexity. Instead, we adopt the ZEKRO protocol [30] to bind the usage of the Edge device signing key to a policy that allows the key to be used only when the Edge is in the correct state. Also, the evidence of each of the children IoT devices is not reported to the External Verifier. Instead, they are verified and kept at the Edge device level.

Key binding: SPARK proposes a means to enforce that the involved parties adhere to the network topology in a privacy-preserving manner. In state-of-the-art swarm attestation schemes for static topologies, a malicious Edge/Aggregator can perform a *"change of path"* attack by substituting one of the IoT attestations with another attestation value from a different IoT device with similar properties (i.e., correct hash of the software/firmware). To the best of our knowledge, SPARK is the first to address this issue as the keys of the children IoT devices are linked to their parent key in such a way that prevents modifications of the attestation result reported by each IoT device.

Tracing: To the best of our knowledge, SPARK is the first swarm attestation scheme to offer a tracing scheme that can be implemented on TPM 2.0 without modifications. In SPARK a trusted Tracer (i.e., the tracing authority) uses a basename to contruct a tracing token. This subsequently enables the Tracer to open (failed) attestation results in order to identify the compromised/affected devices for further diagnostics.

Proof-of-concept (PoC) implementation: We provide a PoC implementation based on a hardware TPM 2.0 and four different embedded devices that closely resemble the capabilities of modern automotive platforms. Using this PoC, we provide an analysis of the device overhead, scalability, network overhead, and tracing overhead. This analysis demonstrates the real-world applicability of SPARK.

2. Related Work

This section discusses related work w.r.t. three domains relevant for this paper: swarm attestation, anonymous signatures, and vehicular networks. Table 1 highlights fundamental differences of SPARK with other swarm attestation schemes.

2.1. Swarm Attestation.

Swarm attestation protocols aim to provide efficient and scalable solutions to attest large networks [6]. Schemes such as SEDA [9], SANA [7], LISA [19], SHeLA [64], FADIA [56] utilize a spanning tree, to efficiently verify the integrity of multiple devices in a large IoT network. In such schemes, the attestation request is propagated through the network hierarchy, with parent nodes relaying it to their child nodes. Subsequently, the attestation result is aggregated at the root node. While these schemes provide efficiency, they are centralized and rely on a single trusted verifier to initiate and verify the attestation result. To overcome the limitations of centralized verification, other swarm attestation schemes such as DIAT [3], US-AID [40], ESDRA [49], and PASTA [47] introduce distributed verification through multiple verifiers. These schemes allow neighbouring devices to mutually attest each other, resulting in improved security. However, one drawback of these schemes is that they do not provide public verifiability of the attestation results.

More recently, swarm attestation schemes have been proposed in literature to enable any party to publicly verify the attestation results. For instance, SCRAPS [63] and PROVE [31] aim to achieve public verifiability in a

TABLE 1: Comparison with state-of-the-art swarm attestation protocols

Scheme	Topology	Provers-Verifiers	Edge devices	TPM 2.0	Publicly verifiable	Evidence Privacy	Tracing*
SEDA [9]	Spanning tree	Many-One	✗	✗	✗	✓	✗
SHeLA [64]	Hierarchy	Many-One	✓	✗	✗	✗	✓
ESDRA [49], US-AID [40], PASTA [47]	Hierarchy	Many-Many	✓	✗	✗	✗	✓
SANA [7]	Spanning tree	Many-Many	✗	✗	✓	✗	✓
SCRAPS [63], PROVE [31]	publish-subscribe	Many-Many	✗	✗	✓	✗	✓
[46], [61]	Hierarchy	Many-One	✗	✗	✗	✗	✓
SPARK	**Hierarchy**	**Many-Many**	✓	✓	✓	✓	✓*

* SPARK proposes a novel tracing scheme that is TPM 2.0-enabled. In this scheme, tracing can only be performed by a certified Tracer authority that is provided with the tracing key.

publish/subscribe IoT network, with subscribers acting as verifiers. In SCRAPS [63], the attestation verification is delegated to a smart contract. In PROVE [31], public verifiability for IoT devices is achieved by leveraging one-way key chains without relying on public-key cryptography.

However, current state-of-the-art swarm attestation schemes do not provide advanced privacy guarantees for swarm devices and do not employ TPM-enabled Edge devices for attestation. In contrast to these existing schemes, SPARK introduces an attestation mechanism that ensures privacy preservation, anonymity, and traceability. In particular, SPARK introduces a novel tracing scheme enabled by TPM 2.0, where tracing can only be conducted by a certified Tracer authority equipped with the tracing key.

2.2. Anonymous Signatures

Anonymous signatures allow a group member to anonymously create signatures on behalf of a group by demonstrating that they possess a valid credential signed by an Issuer without revealing any information about their membership credential or identity. This is typically done via efficient ZKP techniques.

One example of anonymous signatures are ring signatures introduced by Rivest et al. [66]. In ring signatures, the signer takes several public keys (referred to as the ring), and a secret key corresponding to one of the public keys. In general, ring signatures provide full anonymity where the signatures are generated in an unlinkable manner.

Group signatures are another signature type that guarantee that a message was sent by a certified group member without leaking any information about the identity of the group member who signed this message (anonymity) unless an opening authority decides to open the signature (traceability) [16], [21]. Without the tracing key, it should be infeasible for an adversary (even given all the signing keys) to determine the identity of the group member who issued a specific signature. Group signatures have many potential applications, such as trusted computing platforms for protecting users' privacy in public transportation and V2X communication [53].

One key scheme that leverages anonymous signatures is Direct Anonymous Attestation (DAA) [12], [33], [32]. DAA is an attestation protocol that offers anonymity and user-controlled linkability, which is steered by a verifier input called the basename. If a signer uses a fresh or empty basename, the resulting attestations cannot be linked, whereas repeated use of the same basename makes the transactions linkable. The DAA signer consists of a principle signer TPM that creates attestations about the state of the host system, e.g., certifying the boot sequence. These attestations convince a remote verifier that the platform it communicates with is running on top of trusted hardware and using the correct software.

State-of-the-art group signature schemes are not supported by the current TPM 2.0. In this paper, to the best of our knowledge, we present the design and implementation of the first group signature scheme that is fully supported by the TPM 2.0 commands. For this, we propose a novel scheme based on DAA-A, and we show its efficiency in IVN applications, consisting of both ECUs and Zonal Gateways, where all Zonal Gateways are equipped with a TPM as a Root-of-Trust.

2.3. Vehicular Networks

In-Vehicle networks. Previous work has shown that CAN-based vehicular networks lack confidentiality, integrity, authenticity and availability [71]. To safeguard the integrity of IVNs, diverse security solutions have been proposed that can be broadly classified into preventative protection, intrusion detection, authentication and post-protection [43]. One prominent security mechanism in literature is intrusion detection, which can be implemented at various layers of the network stack [38]. However, intrusion detection systems are known to produce false positives and negatives. Hence, another promising security solution is remote attestation, which is more demanding but produces a deterministic result. Various works [46], [61] have proposed centralized attestation of IVNs. In these schemes, a powerful Master ECU runs the attestation of the entire IVN. In VULCAN [73], the authors provide a component attestation of protected modules through a trusted global vehicle attestation server. Furthermore, in [4], the attestation is conducted on an Edge server to which the vehicle connects via an RSU. More recently, a distributed approach has been proposed, where every ECU verifies all the ECUs on which it depends [45]. Nevertheless, these schemes exhibit various limitations, such as high cost, the need for custom hardware, and reliance on network connectivity. Additionally, these schemes do not consider the physical layout of modern vehicular networks and are, therefore, inefficient.

V2X. In the context of V2X, several schemes aim to establish privacy-preserving and/or anonymous authentication [27], [78], [77]. However, these schemes focus on a network of vehicles/infrastructure rather than an IVN. SPARK complements the state of the art by enabling a way to establish trust at the device level between two vehicles.

3. Problem Statement

We consider an IVN that follows the emerging zonal architecture as depicted in Figure 1. In this architecture, the ECUs are grouped based on their physical location within the vehicle. Typically, the ECU firmware contains proprietary information that should remain hidden to third parties [55]. The group of ECUs within one physical zone of the vehicle is managed by a Zonal Gateway and together form a Branch. This Branch is in turn managed by a Central Gateway. Central Gateways include numerous connected interfaces, which can pose a potential attack surface. Consequently, they should remain unaware of the connected ECUs configurations.

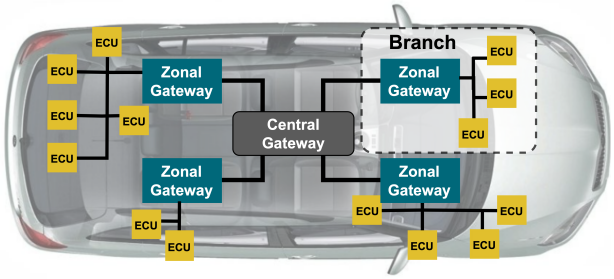

Figure 1: A representation of an In-vehicle Network (IVN) consisting of a Central Gateway, various Zonal Gateways (Edge devices), and lightweight ECUs (IoT devices). The Zonal Gateway together with its children ECUs forms a network branch.

Consider the scenario where a smart vehicle is in motion and some traditional security mechanisms in the IVN, e.g., traffic analysis, notice anomalous behaviour of the connected devices. Due to the possibility of false positives, verifying devices' integrity and tracing the compromised ones would be crucial. However, this is a non-trivial task.

Typically, for lightweight devices, the integrity verification mechanism would require the suspension of the regular operation of the devices. However, given the criticality of the system, it is impractical to suspend the operation of all the devices in the IVN in order to perform the integrity verification and identify the compromised devices. Moreover, IVNs consist of lightweight ECUs with a minimal Trusted Computing Base (TCB) [46], [76], [34], [2] that have limited bandwidth for communication with their respective Zonal Gateways. These Gateways can be subject to tampering. Hence, they are considered untrusted and should not be able to maliciously influence the integrity verification process, e.g., by replacing the results. Furthermore, in a V2X scenario, a vehicle or RSU should be able to verify the integrity of any vehicle in its proximity without prior knowledge of their expected legitimate state. This is because: (1) OEMs keep firmware configurations confidential, and (2) a vehicle cannot feasibly store all firmware variations across all existing IVNs.

This paper aims to address these challenges by leveraging the properties of emerging zonal IVN architectures. In this setting, to ensure the network's integrity, any third-party (internal or external) verifier should be able to verify Zonal Gateways and their connected ECUs. Thus, we consider the Central Gateway as an Internal Verifier that should also be able to relay the attestation result to any External Verifier over V2X technology, such as any other vehicle or a RSU. This approach enables vehicles of different models and brands to verify each other's integrity without revealing any critical information about their IVN configuration. Furthermore, we propose a privacy-preserving tracing mechanism that facilitates a third-party tracer to identify the compromised ECUs (e.g., for replacement/repair).

4. System Model and Threat Model

The system model considers an emerging zonal IVN architecture as presented in Figure 2. Specifically, in our protocol, we consider the presence of the following entities:

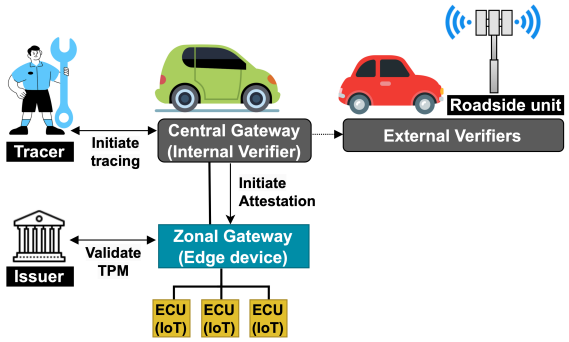

Figure 2: System model in SPARK

ECUs (IoT devices): ECUs are IoT devices. They are potentially untrusted and lightweight embedded devices that interface with the sensors and actuators in a vehicle to control one or multiple electrical subsystems of the vehicle. In line with [46], [76], [34], ECUs devices are equipped with a minimal TCB and have limited processing power and relatively low bandwidth for communication with their respective Zonal Gateways (e.g., $\leq$ 20 Mbit/s for the CAN XL [35]). We further assume that the channel is secured through the CANsec standard [26], which enables encryption and authentication of the data communication. In this system, each ECU is uniquely identified as D_k for $k \in [1, n]$, where n is the total number of ECUs associated with a Zonal Gateway.

Zonal Gateways (Edge devices): Zonal Gateways are powerful embedded devices that act as Edge devices in modern zonal vehicular networks. Since Zonal Gateways can include numerous interfaces, they can be exposed to cyber attacks, and are therefore considered untrusted. They are responsible for managing the ECUs within a single physical zone of the vehicle. This means that each ECU device is only connected to one Zonal Gateway. The Zonal Gateway, along with its connected ECU devices, composes a branch. To communicate with the Zonal Gateway, all the ECUs in a branch share a single communication channel which is a broadcast bus called CAN. Based on [2], [76], each Zonal Gateway is equipped with a Root-of-Trust, such as a TPM 2.0. Zonal Gateways are uniquely identified as Edge_j, for $j \in [1, u]$, where u is the total number of Zonal Gateways in the IVN.

Central Gateways (Internal Verifier): Central Gateways act as the Internal Verifier ($\mathcal{V}_{int}$). Overall, Central

Gateways work as high-end central computation systems that are typically linked to sensors and ECUs via networked Zonal Gateways for processing and consolidation of data. The Central Gateway can identify connected ECUs but should not store their configurations due to privacy, security, and IP concerns. We assume that the channel between Zonal Gateway and the Internal Verifier is implemented using a high bandwidth channel such as AE. Each $\mathcal{V}_{int}$ is equipped with a TPM 2.0.

External Verifier: Any Central Gateway of another vehicle or a V2X infrastructure (e.g., a RSU) acts as an external third-party Verifier ($\mathcal{V}_{ext}$). We assume that $\mathcal{V}_{ext}$ is equipped with a TPM 2.0 and can, therefore, establish a trusted interaction with $\mathcal{V}_{int}$ of the other vehicles.

Issuer: The Issuer (Issuer) is a trusted third-party that validates the TPM of the Zonal Gateway and generates a credential for each of the branches of the IVN. Additionally, the Issuer also validates the TPM of the Central Gateway.

Tracer: The Tracer is an authorized opening authority that can open the attestation results to determine the identity of the compromised device. The tracing process is typically carried out in a certified environment.

To guarantee consistency throughout the remainder of the paper, we assume that our system is established as a zonal architecture. However, in practice, the proposed protocol is designed to be compatible with any network that follows a similar Edge-IoT architecture (including zonal and domain IVN topologies). It is also agnostic to the specific Trusted Component (TC) implementation and can work with TPM or other TC solutions; given the cryptographic operations required by the protocol, other trusted components or legacy platforms capable of performing these operations can ensure its applicability. Furthermore, the protocol can be adapted to utilize different IVN communication protocols, such as FlexRay or AE, provided that a secure and authenticated channel is established in the software.

4.1. Adversarial Assumptions

This subsection provides an intuitive overview of the adversaries that are considered in our system model. In line with [9], [7], [19], we assume a software adversary that aims to compromise ECUs, Zonal Gateways, and Central Gateways ($\mathcal{V}_{int}$). Such an adversary's goal is to forge signatures for ECUs and Zonal Gateways in order to bypass the attestation procedure and evade detection. A software adversary can launch advanced attacks that could compromise the attestation process. One such attack is the *"change of path"* attack, where a malicious Zonal Gateway replaces an ECU's attestation with a different attestation value from another ECU. Another sophisticated attack is the collusion attack, in which multiple ECUs or Zonal Gateways conspire to generate signatures that bypass the attestation mechanism, further compromising the integrity of the system.

Additionally, we consider a communication adversary that attempts to control the communication channel between the ECUs and the Zonal Gateway. Communication adversaries target to spoof, drop, delay, and eavesdrop on the messages. However, w.r.t. the channel between the ECUs and Zonal Gateway, this adversary is inherently hindered by the secure and authenticated channel (e.g., through CANsec) that we assume. Moreover, replay adversaries can send an "old" attestation value to the Zonal or Central Gateway. Additionally, physical adversaries attempt to physically compromise an ECU/Zonal Gateway through replacement or by excluding the device from attestation. Furthermore, in line with [38], this adversary can directly attach to the CAN bus to eavesdrop or spoof on the communication with ultimate goal to forge the attestation.

Furthermore, advanced software attacks, such as runtime attacks, that involve post-boot compromises, are out of our context. However, using the Integrity Measurement Architecture (IMA) [67], our solution can efficiently be adapted to also protect against runtime attacks. Similarly, physical attacks involving compromised/subverted TPMs are also out of scope. Nevertheless, this issue can be addressed with techniques such as [14], [13]. Finally, in line with the state-of-the-art collective attestation schemes [9], [7], [19], we keep the Denial of Service (DoS) attack out of our scope.

5. Security Requirements

In this section, we formally define the following high-level security properties that are achieved by SPARK:

Unforgeability: Given a honest Issuer, no adversary should be able to create a signature on a message m without having access to the TPM or the IoT devices signing keys.

Anonymity: Given two signatures, an adversary should not be able to distinguish whether both signatures were created by one or two different honest branches. The anonymity must hold even if the Issuer is corrupt.

Evidence Privacy: Given a signature σ, an adversary should not learn anything about the devices' exact configuration. All the devices in a branch must provide verifiable evidence to convince an external verifier about the correctness of their state, while not revealing any configuration.

State Correctness: It must be ensured that only authenticated and non-compromised Edge devices can use their embedded TPM to create verifiable evidence of each branch's configuration.

Traceability: It is required that no Edge device or group of Edge devices can generate signatures that cannot be traced back to any of them.

Edge-IoT key Binding: It should be computationally infeasible for an adversary to replace/exclude any of the devices' signatures while the branch attestation is still correctly verified (i.e., *"change of path"* attack). This is a crucial property for swarm topologies as it offers a tight binding relationship between the parent Edge and its children's IoT device keys.

5.1. Device Assumptions

To provide the aforementioned security guarantees we assume that the trusted hardware components provide the following capabilities:

- **TPM (for Edge device).** The TPM of the Edge device can incorporate a variety of checks depending on

the exact application. Typically, a TPM measures the boot of a device. However, using e.g., the IMA, the integrity check can be extended to incorporate post-boot measurements of the device. Hence, SPARK can be efficiently customized to satisfy the security needs of a specific application.

- **Minimal TCB (for IoT device).** In line with the swarm attestation literature and automotive platforms (e.g., SHE or EVITA), the minimal trusted computing base (TCB) includes the following minimal features: (i) A Read-Only Memory (ROM), where integrity-protected attestation code should reside; (ii) A Memory Protection Unit (MPU) that allows to enforce access control on areas of the memory, e.g., read-only access to certain memory areas exclusively to attestation protocol; and (iii) A secure key storage accessible only from the attestation protocol in ROM.

6. Preliminaries

6.1. Notation

Let $\mathbb{F}$ be a finite field and $\tilde{\mathbb{F}}$ denotes a finite extension field of $\mathbb{F}$. Let $\mathbb{E}$ be an elliptic curve defined over $\mathbb{F}$ with a base point G_0. Let $\tilde{\mathbb{E}}$ denote the points of $\mathbb{E}$ over $\tilde{\mathbb{F}}$ with a base point $\tilde{G_0}$.

In our protocol, $\tilde{G_0}$ is used to generate the Issuer's public key in $\tilde{\mathbb{E}}$, whereas G_0 is used to generate the TPM's key in $\mathbb{E}$. The curve $\mathbb{E}$ is equipped with a type III pairing $\tau : \mathbb{E} \times \tilde{\mathbb{E}} \rightarrow \tilde{\mathbb{F}}$. τ is used to verify membership credentials under the Issuer's public key. We define the hash functions: $\mathsf{H} : \{0,1\}^* \rightarrow \mathbb{Z}_q$ and $\mathsf{H_1} : \{0,1\}^* \rightarrow \mathbb{E}$. The operation on $\mathbb{E}$ (resp. $\tilde{\mathbb{E}}$) is written with additive notation. Multiplication by scalars is always written on the left where scalars are elements in $\mathbb{Z}_q$ for a prime number q that represents the order of the subgroup $\langle G_0 \rangle$ in $\mathbb{E}$. An overview of SPARK's notations can be found in Table 2.

6.2. Overview of the DAA-A Protocol

The DAA-A [24] protocol consists of a DAA scheme with an anonymous credential of a public key. But in contrast to a DAA scheme, the public key does not correspond to a single secret key but also multiple attribute keys. By using a single attribute, one gets a protocol that is mathematically identical to the DAA protocol and inherits its security and privacy properties. Two DAA-A schemes are presented in [24], the sDH and the CL-based DAA-A protocols. We adopt the Camenisch-Lysyanskaya (CL)-based DAA-A as it is well suited for our protocol construction because it uses only standard elliptic curve calculations.

7. SPARK Attestation Scheme

In this section, we present the details of our protocol. Overall, the protocol consists of four main phases, namely, (1) Key Setup Phase, (2) Join Phase, (3) Attestation Phase, and (4) Verification Phase. The Setup and Join Phase are one-time procedures, performed offline during the deployment/manufacturing of the vehicle. On the contrary, the Attestation and Verification Phases are performed during the regular operation of the vehicle.

TABLE 2: Notation Summary

Symbol	Description
D_k	The k^{th} IoT device
n	Number of IoT devices in a branch
Edge_j	The j^{th} Edge device
u	Number of Edge devices in the IVN
$\mathcal{V}_{int}$	Internal Verifier
$\mathcal{V}_{ext}$	External Verifier
Issuer	The Issuer
$\mathbb{F}$	A finite base field
$\tilde{\mathbb{F}}$	A finite extension field of $\mathbb{F}$
$\mathbb{E}$	An elliptic curve defined over $\mathbb{F}$
$\tilde{\mathbb{E}}$	The points of $\mathbb{E}$ over the extension field $\tilde{\mathbb{F}}$
q	A prime number that defines the order of cyclic groups
G_0	A basepoint of $\mathbb{E}$
$\tilde{G_0}$	A base point of $\tilde{\mathbb{E}}$
$G, G_1, \ldots, G_n$	public group elements in $\mathbb{E}$
$\tilde{G}, \tilde{G_1}, \ldots, \tilde{G_n}$	public group elements in $\tilde{\mathbb{E}}$
τ	Type III pairing
(x, y)	The Issuer private key
(X, Y)	The Issuer public key
x_T	The Tracer private key
$X_T = x_T G$	The Tracer public key
x_0	TPM's private key
x_k	D_k private key
$X_k = x_k G_k$	D_k public key
$\mathcal{B}$	The branch public key
$(A, B, C, D, E_0, E_1, \ldots, E_n)$	The branch credential created by the Issuer
TK	The TPM's tracing Token
s_k	The IoT signature using its secret key x_k
s_0	The DAA signature of an Edge device using the TPM key x_0
σ	The branch signature
m, ρ	Attestation challenges
H	Hash function defined as $H : \{0,1\}^* \rightarrow \mathbb{Z}_q$
H_1	Hash function defined as $H_1 : \{0,1\}^* \rightarrow \mathbb{E}$

7.1. High-Level Overview of SPARK

This section provides a high-level conceptual overview of the main phases in SPARK without focusing on the cryptographic details. The protocol begins with each branch generating a branch key, effectively binding the ECUs (i.e., IoT devices) to their respective Zonal Gateway (i.e., Edge device). Next, the Issuer certifies the branch key, thereby generating the DAA credential for that branch. Moreover, in this phase, a tracing token associated with the DAA key of the Zonal Gateway's TPM is generated, enabling a trusted Tracer entity to locate compromised branches if necessary. During attestation, the ECUs and zonal gateway interact to produce a branch signature (attestation result) as a single signer, which serves as verifiable evidence of the branch's correct configuration. This signature includes the Zonal Gateway's attestation (i.e., TPM-signed configuration). Additionally, we put partial trust on the Zonal Gateway for not altering the message (ECU configuration) signed by the ECUs. Subsequently, in the verification phase, the Verifier checks the branch signature under the Issuer's public key. In line with other signature schemes (DAA, group signatures, etc.), the Verifier is considered untrusted. Moreover, SPARK enables public verifiability, allowing independent verification without relying on the Central Gateway's firmware trustworthiness. In case of compromise (failed attestation),

the Tracer can identify the compromised Zonal Gateway or ECU by opening the signature.

7.2. Key Setup Phase

The Key Setup Phase is a preliminary procedure that aims to ensure the secure deployment of the protocol in the upcoming phases. Specifically, this phase encompasses the generation of the public group elements and the establishment of the Issuer and IoT keys. To generate the keys for the IoT devices (D_k) associated with an Edge device (Edge_j), we use $G_1, \ldots, G_n \in \mathbb{E}$. Specifically, from the generator $G_0 \in \mathbb{E}$, we generate the public group elements $G_1, \ldots, G_n$ as $G_k = r_k G_0$, where $k = 1 \ldots n$ and n is the total number of IoT devices in a branch. Likewise, we generate $G = r_G G_0$ for use in the generation of the credential by the Issuer. Here, r_G and r_k are taken from $\mathbb{Z}_q$ at random to ensure that there is no known discrete logarithm relation between any two distinct G_k and between G_k and G. As such, the r_G, r_k should be deleted after the Key Setup Phase of our protocol. Similarly, we generate $\tilde{G}, \tilde{G}_1, \ldots, \tilde{G}_n$ from $\tilde{G}_0$ in $\tilde{\mathbb{E}}$ as $\tilde{G} = r_G \tilde{G}_0$ and $\tilde{G}_k = r_k \tilde{G}_0$, respectively.

Every IoT device D_k chooses a random key x_k from $\mathbb{Z}_q$, and calculates the public key $X_k = x_k G_k$ with a proof of construction of X_k. Similarly, the Tracer chooses a random key $x_T \in \mathbb{Z}_q$ with a public key $X_T = x_T G$ with a registered proof of construction. In addition, the signing secret key of the Issuer is composed of two integers $x, y \in \mathbb{Z}_q$, with corresponding public keys $\tilde{X} = x\tilde{G}$ and $\tilde{Y} = y\tilde{G}$. Note that Issuer employs a proof of knowledge, denoted as π^{IPK}, to establish the relationship between the private key components (x, y) and their public key counterparts $(\tilde{X}, \tilde{Y})$. The proof of constructions of the key (a signature of knowledge of the secret part of the key) effectively binds the public key to its corresponding secret key maintaining the protocol's integrity and correctness.

7.3. Join Phase

The Join Phase encompasses the enrollment of the Edge device, the generation of the branch key, the generation of the branch credential, the restriction on the Edge device's signing key and the generation of the tracing token.

In our system, Edge devices are equipped with a TPM. As such, they guarantee the binding of the signing key to the endorsement key of the TPM [23] so that it can be validated by the Issuer. This binding allows the issuer to authenticate the TPM even in the presence of a corrupt host.

After authenticating the TPM, the TPM's key will be certified together with the children IoT devices keys by the Issuer. The issuance of such certification (credential) not only allows the Edge device and its connected IoT devices to start using their keys for attestation services but also binds the Edge device's signing key to its children IoT keys in a way that any change/substitution of one or more keys will result in a failed attestation. This is a crucial requirement for swarms with static topology like IVN applications, where each component's position is fixed by the manufacturer.

7.3.1. Checking eligibility of Edge device. In this step, the Issuer checks whether the Edge device is eligible to join, i.e., the public part of the signing key PK has not been previously certified or revoked. If this validation succeeds, it will compute the credential for the branch.

Internally, the TPM chooses the secret signing key $x_0 \leftarrow \mathbb{Z}_q$ and sets its public key $PK = x_0 G_0$. It returns the PK with a proof of its construction, alongside other parameters (i.e., TPM policy) in an integrity-protected data structure. An authorization session ($request_{CRE}$) is started with the Issuer sending a nonce ρ to the TPM via the host. A registration package can now be assembled, consisting of the key data structure, nonce ρ and the public part of the TPM Endorsement Key (EK). This package is subsequently sent to the Issuer for checking.

7.3.2. Generating the branch key. Here, we generate the branch key, which is a combination of the Edge device's TPM signing key x_0 together with its children IoT keys $x_1, \ldots, x_n$. The construction of the branch key $\mathcal{B}$ binds each Edge device, which is equipped with a hardware key x_0, to its connected IoT devices in a way that if one of the IoT keys x_k is replaced or changed, the whole attestation will fail to prevent "change of path" attack in swarm attestation. First, each IoT child device sends its public key to Edge device. Next, after receiving all its children public keys, the Edge calculates the branch public key $\mathcal{B} = PK + \sum_{k=1}^{n} x_k G_k$, where a branch consists of an Edge device with its children IoT devices. $\mathcal{B}$ will then be certified by the Issuer in the following step.

7.3.3. Computing the branch credential CRE. First, the Edge sends $\mathcal{B}$ to be signed by the TPM with its key x_0 using the EC-Schnorr signature scheme. Then, the Edge forwards the TPM's signature σ_0, $\mathcal{B}$ and PK to the Issuer.

If σ_0 is successfully verified by the Issuer under the TPM's public key PK, if the TPM's key is not certified before and does not match any of the keys in the Revocation List RL, the Issuer then proceeds with the credential creation.

The Issuer chooses a random $t \in_R \mathbb{Z}_q$ as in the CL signature [15], where $\in_R$ denotes uniformly random sampling, and calculates:

$$A = tG,\ B = yA,\ C = xA + txy\mathcal{B},\ D = ty\mathcal{B}$$

$$E_0 = tyG_0 \text{ and } E_k = tyG_k\ \forall\ k \in [1, n]$$

such that $A \neq 1_{\mathbb{E}}$. The Issuer then chooses a random $\gamma \in_R \mathbb{Z}_q$ and calculates the challenge $\hat{c} = H\Big(\gamma G | \gamma G_0 | \ldots | \gamma G_n | \gamma \mathcal{B} | \rho\Big)$ and $\hat{s} = \gamma - \hat{c}ty$, where ρ is a message of freshness agreed by the Issuer and the parent Edge.

The Issuer sends the credential $\mathsf{CRE} = (A, B, C, D, E_0, E_k, \hat{c}, \hat{s})$ back to the Edge device. Upon receiving CRE, the Edge device verifies the signatures under the Issuer's public key $\tilde{X}$ and $\tilde{Y}$ as follows. First, it checks the pairings:

$$\tau(A, \tilde{Y}) \stackrel{?}{=} \tau(B, \tilde{G}) \text{ and } \tau(A + D, \tilde{X}) \stackrel{?}{=} \tau(C, \tilde{G})\,.$$

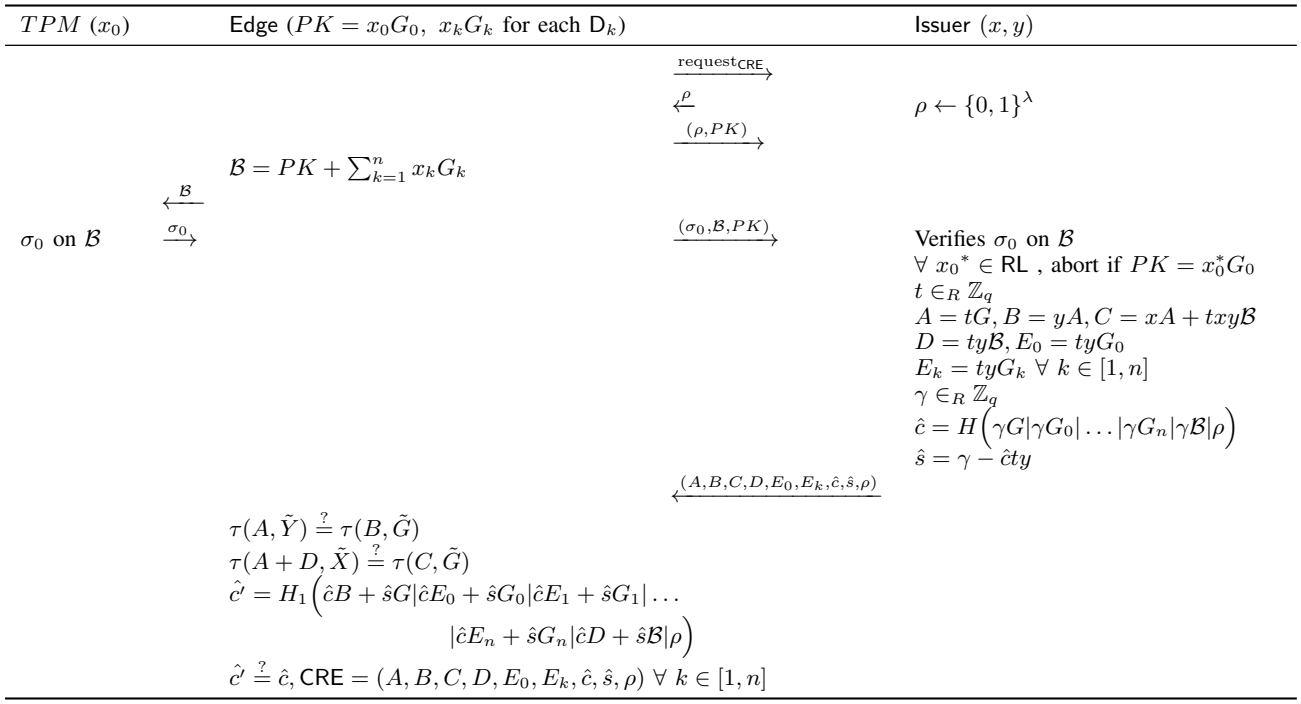

Figure 3: The Join Protocol in SPARK with the Issuer, where the branch credential CRE is issued.

If this check is successful, it then validates the signature $\hat{c'} = H_1\Big(\hat{c}B + \hat{s}G | \hat{c}E_0 + \hat{s}G_0 | \hat{c}E_1 + \hat{s}G_1 | \ldots | \hat{c}E_n + \hat{s}G_n | \hat{c}D + \hat{s}\mathcal{B} | \rho\Big)$ and accepts if $\hat{c'} = \hat{c}$. The final credential CRE is then stored in the parent Edge device database. All of these operations are shown in Figure 3.

7.3.4. Restrict to Trusted Configuration in the Edge device. The TPM includes a set of internal *extendable* registers called Platform Configuration Registers (PCRs) in its non-volatile memory. These PCRs store measurements of the residing Edge as chained hashes originating from a Root-of-Trust for measurements (e.g., CPU microcode, TEEs). It is possible to build a policy that can be satisfied if a selection of PCRs matches a predetermined value, referencing a trusted state. Using `PolicyPCR` [1], we ensure that the Edge device signing key x_0 is inoperable if the integrity of the Edge is compromised.

7.3.5. Achieving traceability. The Tracer is equipped with a key $X_T = x_T G$ for some secret key $x_T \in \mathbb{Z}_q$, with a registered proof of construction. We need a tracing token TK that is associated with the TPM's DAA key to be registered in the Join Phase. Let $bsn_T \in \{0,1\}^*$ denote the Tracer's basename. Each Edge device creates its tracing token using its TPM as follows:

1) The Edge calculates $J_T = H_1(bsn_T) \in \mathbb{E}$ and sends it to the TPM.
2) Using `TPM2_Commit` and `TPM2_Sign`, the TPM chooses a random $\beta \in_R \mathbb{Z}_q$, and calculates $c_T = H(\beta J_T, \beta G_0)$, $s_T = \beta + c_T x_0$ and the tracing token $TK = x_0 J_T$. The TPM sends (s_T, c_T, TK, PK) to the Tracer via the host.
3) The Tracer checks $c_T \stackrel{?}{=} H(s_T J_T - c_T TK, s_T G_0 - c_T PK)$.
4) If the above check is successful, the Tracer adds (TK, PK) to its records.

7.4. Attestation Phase

In this phase, upon receiving an attestation request from the verifier, the actual attestation is conducted. The intuition is that the TPM checks that the device is in the correct state by verifying whether the policy is satisfied. Once the verification is successful, the TPM creates a signature s_0 using its hardware key x_0, and each D_k creates a signature s_k using its key x_k. The Edge device together with its children IoT devices create a proof of knowledge that shows in a privacy-preserving manner, i.e., without leaking any information about the identities of the Edge device or the IoT devices, that the branch key $\mathcal{B} = PK + \sum_{k=1}^{n} x_k G_k$ is certified by the Issuer. This confirms that the branch has a valid credential CRE that can be successfully verified under the Issuer's public key. Moreover, the proof of knowledge provides evidence privacy as revealing the configuration of any of the connected branch devices causes privacy, security, and IP issues. This is in line with design choice to store the IoT configurations at the Edge device level (e.g., in the TPM or in secure memory) rather than at the Internal Verifier level. This approach (1) reduces IVN network overhead by avoiding communication between IoT and the Edge device, and (2) offers sufficient secure storage for the configurations due to multiple Edge devices (each with a TPM). The overall signature σ is verified in a zero-knowledge manner that preserves the privacy of the whole branch (without leaking any information about any of the public keys or CRE). Note that we need the IoT devices to participate in the ZKP to guarantee anonymity of the protocol while achieving the key-binding property to prevent "change of path" attacks. Specifically, $\mathcal{B}$ binds the IoT devices'

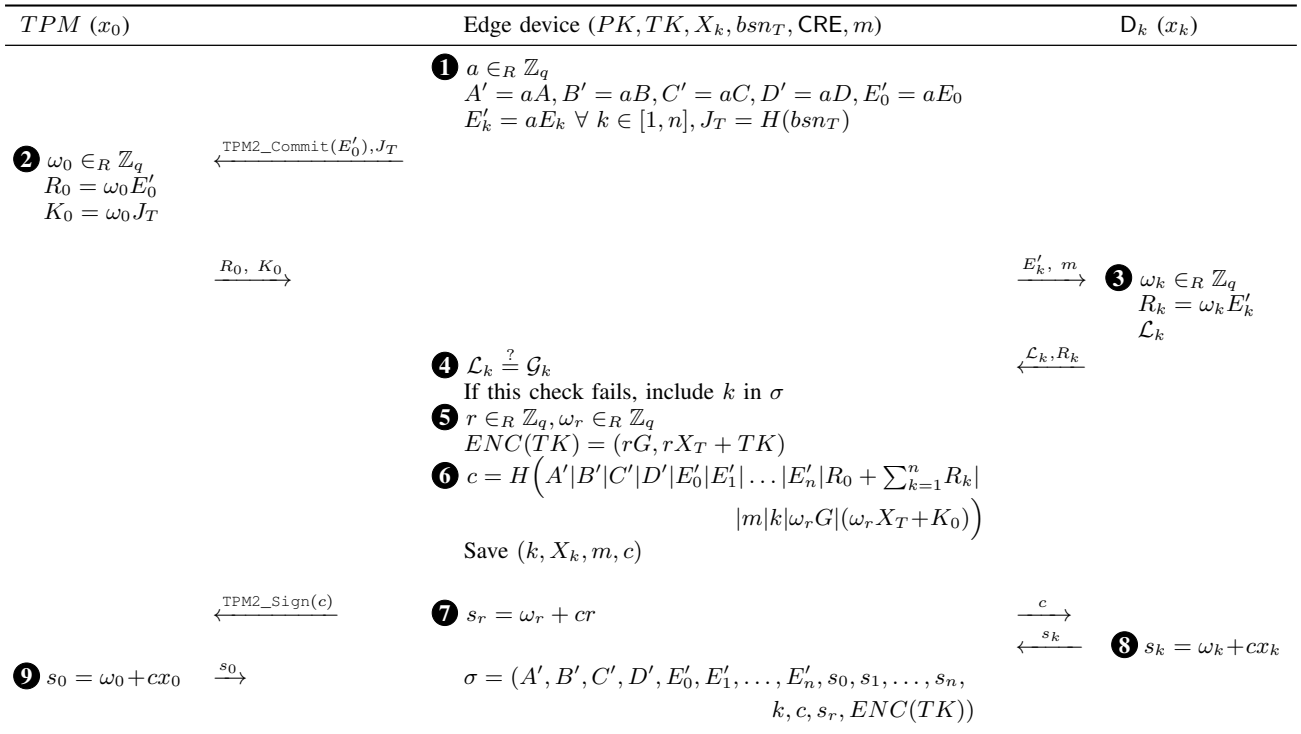

Figure 4: The Attestation Procedure in SPARK describing the interactions between the Edge device with its embedded TPM and the IoT devices.

keys to the TPM key PK in a way that prevents any IoT device from using a key that was not originally used in the creation of $\mathcal{B}$.

The flow of the attestation is presented in Figure 4 and is as follows.

❶ **Randomizing the credential**: The host (Edge) creates a blinding factor $a \in_R \mathbb{Z}_q$. Then it blinds (randomise) the credential CRE by computing: $A' = aA; B' = aB, C' = aC, D' = aD, E'_0 = aE_0, E'_k = aE_k \ \forall \ k \in [1, n]$. The Edge device sends each E'_k to the corresponding IoT device D_k, and E'_0 and $J_T = H(bsn_T)$ to its embedded TPM.

❷ **TPM-Commit**: Using `TPM2_Commit`, the TPM selects a random $\omega_0 \in_R \mathbb{Z}_q$ and calculates $R_0 = \omega_0 E'_0$ and $K_0 = \omega_0 J_T$, the TPM returns R_0 and K_0 to the host. The TPM stores ω_0 in a safe place to be used for the signature generation.

❸ **IoT-Commit and measure**: Each IoT device D_k selects a random ω_k from $\mathbb{Z}_q \ \forall \ k \in [1, n]$, calculates $R_k = \omega_k E'_k$. D_k stores ω_k in a safe place to be used for the branch signature generation. The IoT device D_k further measures its `Current Configuration` $\mathcal{L}_k$, then sends R_k and $\mathcal{L}_k$ to the Edge.

❹ **Evidence check**: The Edge checks $\mathcal{L}_k \stackrel{?}{=} \mathcal{G}_k$ where $\mathcal{G}_k$ represents the `Golden Hash` (i.e., the correct hash of the software/firmware) of the IoT device. If the check fails, the Edge adds the index k of the IoT device in the signature.

❺ **Encryption token generation**: The Edge samples a random $r \in \mathbb{Z}_q$ and calculates the encryption token $ENC(TK) = (rG, V = rX_T + TK)$. To prove the knowledge of r, the Edge then selects a random $\omega_r \in \mathbb{Z}_q$ and calculates the commitment values: $\omega_r G$ and $\omega_r X_T + \omega_0 J_T$. ($\omega_0 J_T$ was sent before from the TPM [step 2: **TPM Commit**])

❻ **Hash**: The Edge calculates the hash value as

$$c = H\Big(A'|B'|C'|D'|E'_0|E'_1|\ldots|E'_n|R_0 + \sum_{k=1}^{n} R_k|m|k|\omega_r G|\omega_r X_T + \omega_0 J_T\Big).$$

The Edge sends c to the TPM and publishes c to the children's IoT devices.

❼ The Edge calculates a signature on r as follows $s_r = \omega_r + cr$.

❽ **IoT-Sign**: The IoT device D_k retrieves the value of ω_k that was used in the **IoT-Commit** phase, then outputs a signature $s_k = \omega_k + cx_k$, and sends s_k to the Edge.

❾ **TPM-Sign**: The TPM retrieves the value of ω_0 that was used in the **TPM-Commit** phase, and outputs a signature $s_0 = \omega_0 + cx_0$ with x_0 as the TPM private key. Please note that a policy restricts the use of the signing key x_0 to the correct configuration of the Edge.

After completing the aforementioned steps, the Edge device sends the overal signature σ to the verifier:

$$\sigma = \Big(A', B', C', D', E'_0, E'_1, \ldots, E'_n, s_0, s_1, \ldots, s_n, c, k, s_r, ENC(TK)\Big).$$

7.5. Verification Phase

In this phase, the verifier verifies the IoT and group signatures. Here, J_T and X_T are assumed to be known by the verifier. The flow of the verification phase is summarized in Figure 5 and is as follows:

① Verify the modified CL certificate by checking the pairings on both the group signature and the credential

constructions:
$\tau(A', \tilde{Y}) \stackrel{?}{=} \tau(B', \tilde{G})$ and $\tau(A' + D', \tilde{X}) \stackrel{?}{=} \tau(C', \tilde{G})$.
② Verify the equivalence of the discrete logarithm using the batch proof trick from [22]: $t_0, t_1, \ldots, t_n \in \mathbb{Z}_q$;

$$\tau(t_0 E'_0 + \ldots + t_n E'_n, \tilde{G}) \stackrel{?}{=} \tau(B', t_0 \tilde{G}_0 + \ldots + t_n \tilde{G}_n)\,.$$

③ Verify the Schnorr ZK proof: to correctly verify the signature, the value of μ should match with $R_0 + \sum R_k$ where $\mu = \sum s_k E'_k + s_0 E'_0 - cD'$.
④ Check the challenge construction

$$c \stackrel{?}{=} H\Big(A'|B'|C'|D'|E'_0|E'_1|\ldots|E'_n|\mu|m|k|s_r G - c(rG)|(s_r X_T + s_0 J_T) - cV\Big)$$

Note that: $\omega_r X_T + \omega_0 J_T = s_r X_T - crX_T + s_0 J_T - cx_0 J_T = (s_r X_T + s_0 J_T) + (-crX_T - cx_0 J_T)$. Here, the first term is known and the second term (which is equal to $-cV$) can be obtained from the Encryption Token. The homomorphic property of ElGamal encryption implies that $ENC(cTK) = cENC(TK)$, which makes the verification work.
⑤ The output is valid if all checks and verification pass.
⑥ If the signature contains one or more index k, the verifier outputs "to be traced / or not".

Edge $(\mathcal{B}, PK, \mathsf{CRE})$	Verifer $(\tilde{X}, \tilde{Y}, bsn_T, X_T)$
$\xleftarrow{m}$	
$\xrightarrow{\sigma, m}$	$\tau(A', \tilde{Y}) \stackrel{?}{=} \tau(B', \tilde{G}_0)$
	$\tau(A' + D', \tilde{X}) \stackrel{?}{=} \tau(C', \tilde{G}_0)$
	$t_0, t_1, \ldots, t_n \in_R \mathbb{Z}$
	$\tau(t_0 E'_0 + t_1 E_1 + \ldots + t_n E'_n, \tilde{G}) \stackrel{?}{=} \tau(B', t_0 \tilde{G}_0 + \ldots + t_n \tilde{G}_n)$
	$\mu = \sum s_k E'_k + s_0 E'_0 - cD'$
	$c \stackrel{?}{=} H\Big(A'\|B'\|C'\|D'\|E'_0\|E'_1\|\ldots\|E'_n\|\mu\|m\|k\|s_r G - c(rG)\|(s_r X_T + s_0 J_T) - cV\Big)$

Figure 5: The Verification procedure in SPARK

7.6. Tracing

Tracing in SPARK allows authorized Tracers to identify compromised devices. This further enables the revocation of compromised devices. Please note that traceability is an optional procedure that can be deactivated from the protocol, reducing the overall latency of the protocol. Tracing can be achieved in the following manner.

1) For the Tracer to trace, it verifies the signature, then decrypts $ENC(TK)$ using x_T and retrieves TK as follows: $TK = V - x_T(rG)$. Note that this encryption is Chosen Plaintext Attack (CPA) secure but not Chosen Ciphertext Attack (CCA) secure. However, this can be easily achieved using the techniques such as in [74].
2) The Tracer outputs the PK that corresponds to TK from its records.
3) A trusted revocation authority sends an "authenticated" request to the Tracer that opens the signature and sends back PK. The revocation authority then adds PK in the Revocation List $\mathbb{RL}$ and its corresponding credential will be revoked.

7.7. External Verification

We assume that both $\mathcal{V}_{int}$ and $\mathcal{V}_{ext}$ are equipped with a TPM. Hence, we consider a trust relationship between $\mathcal{V}_{int}$ and $\mathcal{V}_{ext}$. This implies that typically $\mathcal{V}_{ext}$ does not verify the attestation result but queries $\mathcal{V}_{int}$ for its most recent attestation result. Whenever critical data has to be communicated with another vehicle or an RSU, the external party first checks the timestamp of the last attestation. However, if there is no recent attestation, $\mathcal{V}_{ext}$ initiates the attestation of the branches by communicating through $\mathcal{V}_{int}$, which relays the requests/responses. Only if the attestation is successful, the external party will accept data from the vehicle.

8. Security Analysis

In this section, we provide a proof sketch to show that SPARK satisfies the security requirements described in Section 5. Our traceability and anonymity games are based on the security model defined in [37]. Formal definitions of the unforgeability, anonymity, and traceability of group signatures are presented in Appendix B.

Theorem 8.1. *SPARK is unforgeable if the DAA-A scheme is EUF-CMA (Existential Unforgeability under Chosen Message Attack) secure.*

To show that SPARK is unforgeable, we need to prove that the advantage of an adversary $\mathcal{A}$ in attacking the unforgeability is negligible. This property should hold even for branch devices (IoT devices and their corresponding parent Edge device) that collude to produce a forged signature. As in DAA-A, the unforgeability of the credential CRE is based on the unforgeability of the CL signature [15]. This signature scheme is existentially unforgeable against a chosen-message attack (EUF-CMA) under the LRSW assumption [54]. The unforgeability of the CL signature (credential) requires the Issuer to be honest. The unforgeability of the signature requires an adversary to output a valid forgery σ^* on a chosen message m^* without knowing the keys and after several signing queries, but the message m^* has not been queried for signing. We argue that our unforgeability is based on the hardness of the Discrete Logarithm (DL) problem.

The main idea of the proof is to have the adversary output two distinct forgeries corresponding to the same random oracle query but for two distinct answers which enables the computation of the discrete logarithm of the public key using the Forking Lemma [68]. Suppose that an adversary $\mathcal{A}$ outputs a valid forgery on the branch's signature with a probability ε using a random secret $\alpha = \{\alpha_0, \alpha_1, \ldots \alpha_n\}$. $\mathcal{A}$ outputs $s^*_0, \ldots, s^*_n, c^*$ as a part of SPARK signature, such each s^*_i is a Shnorr signature using the secret α_i. There exists a simulator that can program the underlying random oracle and outputs the witness α with probability $\frac{\varepsilon^2}{q_A}$, where q_A is the number of random oracle queries.

Theorem 8.2. *The anonymity of SPARK requires the honesty of the branch-connected devices, it follows from the anonymity of the DAA-A scheme in [24] and the CPA security of the ElGamal encryption scheme.*

Assume that an adversary $\mathcal{A}$ is given a signature σ generated by using a key $x_{0_{i_b}}$, where b can be either 0 or 1 that represents two different identities i_0 or i_1, respectively. To demonstrate the anonymity of SPARK, we show that $\mathcal{A}$ outputs the correct identity i_b, the identity of the group member with a signing key $x_{0_{i_b}}$, only by a random guess of b with a success probability $\frac{1}{2}$. In our protocol, the identity of each Edge device is protected/hidden by performing the attestations using the ZKPs about the validation of the credential without revealing any information about the group member's public key or the credential. The anonymity of the IoT devices is only achieved against external verifiers due to the Zero-Knowledge property of the Schnorr proof of knowledge. This property allows the verifier to learn nothing about the keys or the identities but only the fact the keys are well certified in the Join Phase. In SPARK, each IoT device generates a Schnorr signature that does not reveal any of the following details: (1) the identity of IoT device, (2) the IoT's public key X_k, or (3) the branch credential CRE. Additionally, note that due to the CPA security of the ELGamal encryption scheme, the encryption token $ENC(TK)$ does not reveal any information about the actual TK. This is due to the randomness r used to generate the encryption token, which preserves the anonymity of the Edge device against any external verifier.

Theorem 8.3. *SPARK ensures correctness and privacy of the Edge device configurations.*

Before communicating with the Issuer in the SPARK join phase, the Edge device computes a policy for the upcoming key that only makes the key usable if the policy stored in the Policy Index (PI) is satisfied. It takes the unique index name $\mathcal{N}$ and computes the policy digest according to the TPM standards [1]. The Edge sends it to the TPM with instructions to generate a new key under the newly created policy. The TPM then creates its signing key $x_0 \leftarrow \mathbb{Z}_q$ and the corresponding public key $PK = x_0 G_0$. It returns PK, along with other parameters (i.e., the policy) in an integrity-protected data structure. Subsequently, an authorization session is started with the TPM, returning a nonce n to the Edge. A registration package consisting of PK, nonce n, index name $\mathcal{N}$, and the public TPM Endorsement Key (EK) is then sent to the Issuer. The Issuer ensures that the contents of the PI are satisfied as a policy, and computes the key-restriction policy that is written to the PI. This policy, $\mathcal{K}$, can only be satisfied by proof from the Edge device showing that the integrity of the Edge is not compromised. The Issuer signs $\mathcal{K}$ and certifies PK as in Section 7. This clever key restriction policy allows the TPM key to be used only if the Edge device is in its correct state. There is no need in SPARK to reveal the Edge device configuration to any external verifier (evidence privacy), yet a successful attestation issued from the Edge device proves that the Edge is in its correct state (correctness).

Theorem 8.4. *SPARK always traces back to the correct Edge device identity due to the correctness of the ElGamal encryption scheme and the unforgeability of the CL signature scheme.*

We argue that given a valid signature, our traceability algorithm always traces back to the correct Edge device's identity. This property should hold even in the scenario of colluding Edge devices from distinct branches. This is due to the correctness of the adopted ElGamal scheme, i.e., using the identified Tracer's key x_T, an honest Tracer will be able to retrieve the correct TK and the corresponding PK relying on its records.

Traceability means that it should be infeasible for an adversary $\mathcal{A}$ who corrupts some set $\mathcal{C}$ of users to output a valid signature that cannot be traced to any member of $\mathcal{C}$. Suppose that $\mathcal{A}$ in our protocol corrupts a set $\mathcal{C}$ of Edge devices, then the adversary outputs a valid signature σ using x_0^i for some $i \in \mathcal{C}$. The signature $\sigma = (A', B', C', D', E_0', E_1', \ldots, E_n', s_0, s_1, \ldots, s_n, k, c, s_r, ENC(TK))$ should be valid in order to be traced. If this signature opens to an identity $i^* \notin \mathcal{C}$, then we have s_0 is constructed using two distinct keys $x_0^i \neq x_0^{i^*}$. This contradicts the soundness of the Schnorr signature of knowledge in which a valid signature (s_T, c_T) is generated using a unique discrete logarithm solution of PK and TK that is known to the prover. Hence i^* should match to one $i \in \mathcal{C}$ since in our scheme the Issuer checks that different group members are not sharing the same keys in the Join Phase. Thus, SPARK's construction ensures that no multiple x_0 values match one signature which results in correct traceability. Therefore, $\mathcal{A}$ will abort with a probability of $1 - \frac{1}{N} + \varepsilon(\lambda)$, where N represents the group size, λ is a security parameter, and $\varepsilon(\lambda)$ is a negligible function that arises from the possibility that $\mathcal{A}$ abuses the soundness of the ZKPs.

Suppose that $\mathcal{A}$ wants to output a "modified" signature that does not trace back to the TPM that created it using its certified key x_0. If $\mathcal{A}$ replaces the TPM's signature s_0 with s_0^* that is generated by a key $x_0^* \neq x_0$, and replaces V with V^* such that V^* traces to x_0^*, then this modification leads to a failed verification. Modifying s_0 would not allow the verification of the ZKP of the credential CRE that is associated with x_0 to pass. The only way for $\mathcal{A}$ to pass the proof is to forge a credential on x_0^*, but this breaks the unforgeability of the CL signature. Whenever an IoT device generates a signature, the parent Edge device (which is assumed to be trusted in tracing its children IoT devices) will be able to trace back the identity of the IoT device, relying on its records that list (k, X_k, m, c).

Theorem 8.5. *SPARK satisfies Edge-IoT key Binding property, i.e. It is computationally infeasible for an adversary to replace/exclude any of the IoT devices' signatures while the branch attestation is still correctly verified.*

Suppose that an adversary $\mathcal{A}$ receives SPARK signature σ from an Edge device. The $\mathcal{A}$ modifies the output of one of the IoT devices D_i, with a signing key x_i. $\mathcal{A}$ replaces the signature s_i by s_i^* that is issued from a corrupt IoT device D_i^* that is not a child of the Edge device. $\mathcal{A}$ calculates $s_i^* = \omega_i^* + c x_i^*$ for a random ω_i^* and using a key $x_i^* \neq x_i$ for the challenge c retrieved from σ. We argue that the signature will not be correctly verified.

Note that the pairings in the verification in Figure 7 will still pass as the Edge device's contribution in σ is not modified. If a modification of the Edge device's signature part is applied whilst σ still correctly verifies, then this breaks the unforgeability of the DAA-A signature. After checking the pairings, the verifier calculates

$\mu^* = \sum s_1 E'_1 + \ldots s_{i-1} E'_{i-1} + s^*_i E'_i + s_{i+1} E'_{i+1} + \cdots + s_n E'_n + s_0 E'_0 - cD'$. The verifier then checks the challenge: $c \stackrel{?}{=} H\Big(A'|B'|C'|D'|E'_0|E'_1|\ldots|E'_n|\mu^*|m|k|s_rG - c(rG)|(s_rX_T + s_0J_T) - cV\Big)$.

σ verifies correctly if and only if $\mu^* = R_0 + \sum_{k=1}^{n} R_k$. Recall that if the above equation verifies then every individual signature s_k and s_0 should verify correctly. This is because r_k is randomly chosen from $\mathbb{Z}_q$ in the setup phase to ensure that there is no known discrete logarithm relation between any two distinct G_k and between G_k and G. Therefore, $s_k E'_k - cx_k E'_k = \omega_k E'_k = R_k$ for all IoT signatures in σ. In particular $s^*_i E'_i - cx_i E'_i = \omega_i E'_i = R_i$ must be satisfied, but this is only valid when $x^*_i = x_i$ due to the unforgeability of the Shnorr signature. This contradicts the assumption that the $\mathcal{A}$ chooses x^*_i with $x^*_i \neq x_i$. Therefore $\mathcal{A}$ can only pass if $\mathcal{A}$ guesses $x^*_i = x_i$, this would happen with a probability of $1/p$ which is negligible for a large p. If $\mathcal{A}$ excludes one of the IoT signatures $s_i \neq 0$ from σ, then $\mathcal{A}$ only succeed if D_i has randomly chosen ω_i such that $\omega_i E'_i = R_i = cx_i E'_i$, if σ verifies correctly then $\omega_i = cx_i$ since $E'_i \neq 0$, this yields to $s_i = 0$ and contradicts our assumption.

9. PoC Implementation

This section assesses the real-world applicability of the SPARK protocol by developing a PoC implementation and by evaluating its performance in terms of efficiency and scalability using representative hardware platforms. Please note that, although implemented, we do not explicitly elaborate on the `Join` phase. This is because the `Join` phase is a one-time process that can typically be executed in a secure environment (e.g., during the manufacturing phase).

For our PoC implementation, we commenced with the DAA implementation of [75] as our initial framework. The reference implementation was developed in C++ using the GNU GCC compiler [36], IBM TSS software stack [39], OpenSSL [62], and the Apache Milagro crypto library (AMCL) [8]. Subsequently, for our PoC, we enhanced the framework to operate in 64-bit ARMv8 mode and updated the framework to use the more recent MIRACL library [58] for the pairing operations. Testing and benchmarking of our PoC were conducted by simulating one network branch on a Raspberry Pi 4, interfacing with an Infineon Optiga™ SLB 9670 TPM 2.0 through the Linux TPM device driver. All elliptic curve operations were performed along the pairing-friendly BN_P256 curve and SHA-256 was taken as a hash function. Since these algorithms are mandatory as per the TPM 2.0 specification, our implementation can be generalized to work with all TPM 2.0 devices.

Additionally, we provide an analysis of the performance implications associated with cryptographic operations on ECUs. This analysis is particularly relevant for an IVN setting, given the inherent constraints of the automotive ECUs in terms of computational power and storage capacity. The cryptographic operations of the ECUs consist of: (1) the generation of a random number, (2) a point multiplication, (3) a SHA-256 hash over the firmware, and (4) the computation of a signature. To evaluate the performance of these operations, we use three lightweight embedded platforms that closely resemble modern automotive ECUs [65], [42], [60], [70]: an Arduino Uno R4 Minima board running a RA4M1 (Arm Cortex M4) operating at 48 MHz, a Raspberry Pi Pico board running a RP2040 (Arm Cortex M0+) operating at 125 MHz, and an Arduino Nano ESP32 board running a NORA-W106 (ESP32-S3) operating at 240 MHz. In our experiments, we use the MIRACL library to implement our cryptographic operations.

10. Evaluation Results

10.1. Overhead on the Devices

In this section, we provide an evaluation on the device-level overhead imposed by SPARK. We separate our discussion according to the ECU and Zonal Gateway overhead.

ECUs. To evaluate the overhead on the ECUs during the `Attestation` phase, we measure the execution time of the cryptographic operations over 1000 runs. Additionally, we compare the memory footprint of our evaluation program, which includes the necessary library functions, to an empty Arduino program. By doing so, we are able to determine the percentage of total program memory that is occupied by cryptographic functionalities on a given platform. Our findings are presented in Table 3.

The results demonstrate that overhead on the ECUs during the `Attestation` phase is kept to a minimum. For the case of one block of firmware, the Arduino Uno, Raspberry Pi Pico, and Arduino Nano devices require 146.234 ms, 138.944 ms, and 34.411 ms of processing time, respectively. It is important to note that random numbers were generated using a slow software-based RNG included in the MIRACL library.

Zonal Gateway. To evaluate the scalability of SPARK during the `Attestation` and `Verification` phase, we measure the execution time of our protocol averaged over 100 runs with a network size varying from 1 to 1024 ECUs. Specifically, in our PoC, we consider the following settings:

- We strip the protocol of TPM policies. This is because the policies are independent of the network size and are thus less relevant for this analysis.
- We consider a benign network, eliminating the need for tracing since this is an optional procedure. A detailed evaluation of the tracing is provided in Section 10.3.
- We assume that all the ECUs can operate simultaneously, while the operations on the Zonal Gateway (described in Sec. 7.4) happen sequentially. This implies that the Zonal Gateway waits for the responses of all ECUs before proceeding with the next step. For the ECU timings, we consider the timings presented in Table 3. Moreover, our PoC does not consider network communication and the involved serialization.
- We perform the verification on the same hardware platform used for the Zonal Gateway. In practice, the Central Gateway will be more powerful than the Zonal Gateway, resulting in improved timing.

The results of the scalability analysis are presented in Figures 6 and 7 for the `Attestation` and

TABLE 3: Overhead of the cryptographic operations performed on the ECUs devices during the `Attestation` phase. The results are evaluated for 3 different automotive-representative embedded platforms.

	Arduino Uno (Cortex M4)		**Raspberry Pi Pico (M0+)**		**Arduino Nano (ESP32)**	
	Mean (ms)	Std (ms)	Mean (ms)	Std (ms)	Mean (ms)	Std (ms)
Generate random	10.611	0.031	5.641	0.017	2.152	0.006
Point multiplication	134.920	0.403	132.846	0.452	31.907	0.100
SHA-256 hash (2 blocks)	0.320	0.001	0.212	0.018	0.141	0.016
Signature	0.383	0.017	0.245	0.023	0.211	0.018
Program memory overhead	13028 (bytes)	4.97 %	14528 (bytes)	0.69 %	23900 (bytes)	0.76 %

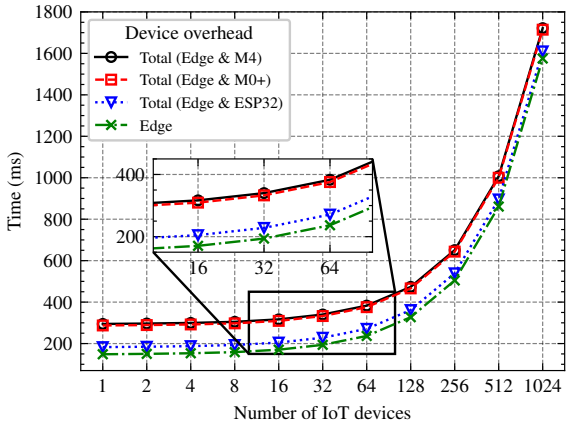

Figure 6: Device-level overhead during `Attestation` demonstrating the Zonal Gateway overhead and the total protocol overhead considering different ECU platforms. The results are highlighted for typical branch sizes around 32 ECUs. The X-axis is of a logarithmic nature.

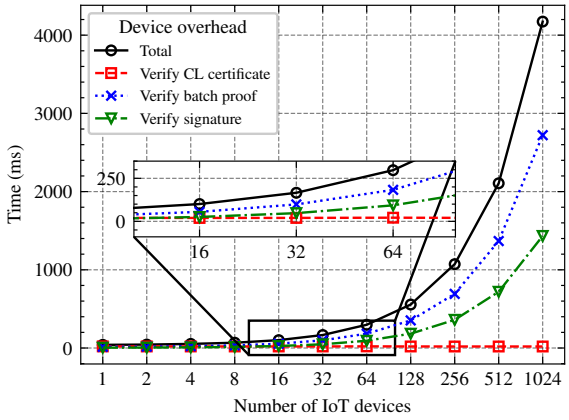

Figure 7: Device-level overhead during the `Verification` procedure depicting the overhead for the different verification steps. The results are highlighted for typical branch sizes around 32 ECUs. The X-axis is of a logarithmic nature.

`Verification` phase, respectively. Based on the scalability analysis, we conclude that the attestation procedure for a typical CAN network branch of 32 ECUs[1] introduces an overhead of 193.81 ms on the Zonal Gateway. Moreover, the total attestation overhead for this network is found to be 340.04 ms, 332.75 ms, and 228.22 ms, for the Arduino Uno, Raspberry Pi Pico, and Arduino Nano as ECUs, respectively. Furthermore, the results demonstrate that the verification phase, which involves complex pairing operations, can be executed within 166.08 ms for 32 IoT devices on a Raspberry Pi 4, indicating the real-world applicability of SPARK.

Timing distribution. Here, we provide an analysis of how the timings are distributed between the Zonal Gateway (Edge and TPM), and the ECUs during the `Attestation` phase. The results of the timing distribution are depicted in Figure 8. For our analysis, we assume the same experimental settings as in our scalability experiment, and we consider the Arduino Nano as ECU platform. Our analysis demonstrates that for a small number of ECUs, the TPM commands constitute the largest component in the attestation overhead. However, for a larger number of ECUs, the operations on the Edge platform become the predominant factor. Specifically, for the case of 32 ECUs, we find an overhead of 53.20 ms, 140.60 ms, and 34.41 ms on the Edge, TPM and ECU platforms, respectively. This also implies that our protocol is more efficient for vehicles with a small number of large branches than vehicles with many small branches. Additionally, our timing distribution analysis highlights that most of the overhead occurs on the TPM. Thereby SPARK does not infer with the real-time operation of the ECU and Zonal Gateway.

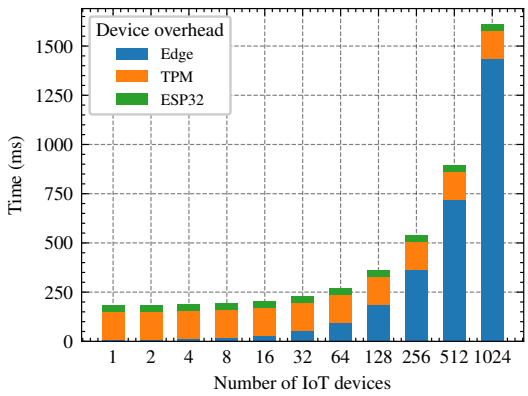

Figure 8: Breakdown of the timing distribution between the Zonal Gateway (Edge and TPM), and ECUs during the `Attestation` procedure considering the ESP32 as the ECU platform. The X-axis is of a logarithmic nature.

10.2. Network Overhead Analysis

In this section, we detail the overhead of our protocol on the network resources. Since we assume that

1. We consider 32 as a typical maximum branch size in modern automotive networks.

the communication between Zonal and Central Gateway is implemented using a high-bandwidth channel (e.g., 100Base-T1 AE), we only detail the communication between Zonal Gateway and the ECUs. In Appendix C, we derive the per-ECU overhead on the network considering the worst-case transmission delays for CAN, CAN FD, and CAN XL. Furthermore, we assume the maximum transmission speeds of the CAN standards. Our results demonstrate that for the considered branch size of 32 ECUs, the overhead on the network is limited to 120.960 ms, 14.740 ms, and 8.7184 ms for CAN, CAN FD, and CAN XL, respectively. This corresponds to bus load increments of 12.09%, 1.47%, and 0.871%, respectively. It is important to note that the increase in bandwidth only applies during the attestation process. As such, SPARK does not compromise the normal operation of the vehicle. Moreover, we provide a discussion of SPARK's real-world applicability in Section 11.

10.3. Tracing Overhead Analysis

In this section, we evaluate the efficiency and effectiveness of our protocol in the presence of a compromised network. To do so, we repeat the experiments conducted in Section 10.1 while introducing a malicious ECU in our simulated network branch and enabling tracing. The findings from the tracing analysis are illustrated in Figure 9.

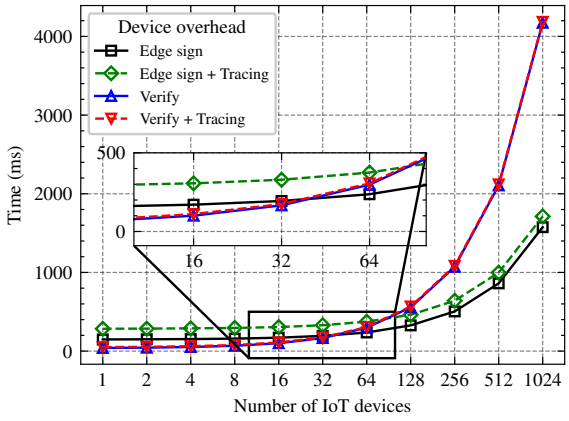

Figure 9: Device-level tracing overhead comparing the Zonal Gateway's `Attestation` overhead and the Central Gateway's `Verification` overhead with and without tracing enabled. The results are highlighted for typical branch sizes around 32 ECUs. The X-axis is of a logarithmic nature.

Our analysis shows that the tracing overhead for a network of 32 ECUs with one malicious ECU is limited to 136.04 ms for attestation and 8.16 ms for verification. Additionally, we note that the majority of the tracing overhead is caused by the `TPM2_commit` command. This is because the `TPM2_commit` algorithm requires more steps when using a basename [1] as required for tracing. Furthermore, this implies that the number of malicious ECUs and the size of the network do not have a significant impact on the tracing overhead. We refer the interested reader to Appendix D for a detailed summary of the device-level overhead in SPARK with and without tracing.

11. Discussion

In SPARK, we proposed a swarm attestation protocol that attests in a privacy-preserving manner the integrity of IVNs that follow the Zonal architecture. However, SPARK is not limited to this specific domain, as discussed below.

Implementation on legacy IVNs. SPARK is backwards compatible with older CAN standards and domain architectures. However, establishing an authenticated and encrypted channel between the ECUs and the Zonal Gateway on legacy CAN networks without dedicated hardware support may be challenging, as it requires software implementation of the symmetric algorithms. This can put additional overhead on the resource-constrained ECUs.

Real-world applicability. As shown in Section 9, SPARK is lightweight. However, we note that a major factor in the attestation overhead are the TPM commands that are performed once per branch. Hence, our solution is more efficient for large branches than for multiple small branches.

Application in other domains. SPARK's applicability extends beyond IVN domains, encompassing diverse fields such as healthcare, wind energy, and industrial automation, to name a few. This is because the cryptographic operation of SPARK is agnostic of the communication protocol. Hence, SPARK is compatible with all static swarms that exhibit an IoT-Edge-Verifier structure.

12. Conclusion

In this paper, we propose SPARK, a novel swarm attestation protocol that aims to guarantee privacy and anonymity and enables third-party verification of the swarm. SPARK relies on a novel group signature scheme that is fully implemented on standardized TPM 2.0. By relying on TPM 2.0, SPARK enables key binding, which enforces that the devices in the network are bound to their physical network topology. Additionally, the proposed protocol provides an efficient tracing solution that allows an opener to trace the compromised device (e.g., for maintenance or replacement). Moreover, in this paper, we provide a representative implementation demonstrating the applicability and efficiency of SPARK in a real-world setting involving IVNs.

As future work, we will explore the opportunities of SPARK in a real-world V2X setting. Additionally, we will explore state-of-the-art attestation mechanisms to provide solutions for runtime attacks and enable interruptibility in SPARK.

Acknowledgment

Wouter Hellemans is an SB Ph.D. fellow at FWO (Research Foundation Flanders) under grant agreement 1SH3824N. Additionally, this work is partially supported by Cybersecurity Initiative Flanders (VR20192203). Nada El Kassem and Liqun Chen would like to thank the European Union's Horizon research and innovation program for support under grant agreement numbers: 101069688 (CONNECT), 101070627 (REWIRE) and 101095634 (ENTRUST). These projects are funded by the UK government's Horizon Europe guarantee and administered by UKRI.

References

[1] Trusted Platform Module Library Part 1: Architecture. Standard, Trusted Computing Group (TCG), 2016.

[2] TCG TPM 2.0 Automotive Thin Profile For TPM Family 2.0; Level 0. Standard, Trusted Computing Group (TCG), 2018.

[3] Tigist Abera, Raad Bahmani, Ferdinand Brasser, Ahmad Ibrahim, Ahmad-Reza Sadeghi, and Matthias Schunter. DIAT: Data Integrity Attestation for Resilient Collaboration of Autonomous System. In *26th Annual Network & Distributed System Security Symposium (NDSS)*, 2019.

[4] Tejasvi Alladi, Sombuddha Chakravarty, Vinay Chamola, and Mohsen Guizani. A lightweight authentication and attestation scheme for in-transit vehicles in iov scenario. *IEEE Transactions on Vehicular Technology*, 69(12):14188–14197, 2020.

[5] Onur Alparslan, Shin'ichi Arakawa, and Masayuki Murata. Next generation intra-vehicle backbone network architectures. In *2021 IEEE 22nd International Conference on High Performance Switching and Routing (HPSR)*, pages 1–7. IEEE, 2021.

[6] M. Ambrosin, M. Conti, R. Lazzeretti, M. M. Rabbani, and S. Ranise. Collective remote attestation at the internet of things scale: State-of-the-art and future challenges. *IEEE Communications Surveys Tutorials*, 22(4):2447–2461, 2020.

[7] Moreno Ambrosin, Mauro Conti, Ahmad Ibrahim, Gregory Neven, Ahmad-Reza Sadeghi, and Matthias Schunter. SANA: Secure and Scalable Aggregate Network Attestation. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security - CCS '16*, 2016.

[8] Apache Milagro. The Apache Milagro Cryptographic Library. https://github.com/miracl/amcl. [Online; accessed 28-August-2024].

[9] N Asokan, Ferdinand Brasser, Ahmad Ibrahim, Ahmad-Reza Sadeghi, Matthias Schunter, Gene Tsudik, and Christian Wachsmann. SEDA: Scalable Embedded Device Attestation. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, pages 964–975, 2015.

[10] Aldin Berisa, Adis Panjevic, Imran Kovac, Hans Lyngbäck, Mohammad Ashjaei, Masoud Daneshtalab, Mikael Sjödin, and Saad Mubeen. Comparative evaluation of various generations of controller area network based on timing analysis. In *2023 IEEE 28th International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1–8. IEEE, 2023.

[11] Unmesh D Bordoloi and Soheil Samii. The frame packing problem for can-fd. In *2014 IEEE Real-Time Systems Symposium*, pages 284–293. IEEE, 2014.

[12] Ernie Brickell, Jan Camenisch, and Liqun Chen. Direct anonymous attestation. In *Proceedings of the 11th ACM conference on Computer and communications security*, pages 132–145, 2004.

[13] Jan Camenisch, Liqun Chen, Manu Drijvers, Anja Lehmann, David Novick, and Rainer Urian. One tpm to bind them all: Fixing tpm 2.0 for provably secure anonymous attestation. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 901–920. IEEE, 2017.

[14] Jan Camenisch, Manu Drijvers, and Anja Lehmann. Anonymous attestation with subverted tpms. In *Annual International Cryptology Conference*, pages 427–461. Springer, 2017.

[15] Jan Camenisch and Anna Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In *Advances in Cryptology—EUROCRYPT 2001: International Conference on the Theory and Application of Cryptographic Techniques, 2001*, pages 93–118. Springer, 2001.

[16] Jan Camenisch and Markus Stadler. Efficient group signature schemes for large groups. In *Annual international cryptology conference*, pages 410–424. Springer, 1997.

[17] CAN specification. Standard, Robert Bosch GmbH, Stuttgart, 1991.

[18] CAN with flexible data-rate specification. Standard, Robert Bosch GmbH, Stuttgart, 2012.

[19] Xavier Carpent, Karim El Defrawy, Norrathep Rattanavipanon, and Gene Tsudik. LIghtweight Swarm Attestation: a Tale of Two LISA-s. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, ASIACCS '17, pages 86–100. ACM, 2017.

[20] Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, Stefan Savage, Karl Koscher, Alexei Czeskis, Franziska Roesner, and Tadayoshi Kohno. Comprehensive experimental analyses of automotive attack surfaces. In *20th USENIX security symposium (USENIX Security 11)*, 2011.

[21] Lidong Chen and Torben P Pedersen. New group signature schemes. In *Workshop on the Theory and Application of of Cryptographic Techniques*, pages 171–181. Springer, 1994.

[22] Liqun Chen. A daa scheme using batch proof and verification. *TRUST*, 10:166–180, 2010.

[23] Liqun Chen, Nada El Kassem, and Christopher J P Newton. How To Bind A TPM's Attestation Keys With Its Endorsement Key. *The Computer Journal*, 2023.

[24] Liqun Chen and Rainer Urian. Daa-a: Direct anonymous attestation with attributes. In *Trust and Trustworthy Computing: 8th International Conference, TRUST*, pages 228–245. Springer, 2015.

[25] Controller area network extra long (CAN XL). Standard, CAN in Automation (CiA), Nuremberg, 2022.

[26] CAN XL add-on services - Part 2: Security. Standard, CAN in Automation (CiA), Nuremberg, 2022.

[27] Jie Cui, Jing Zhang, Hong Zhong, and Yan Xu. Spacf: A secure privacy-preserving authentication scheme for vanet with cuckoo filter. *IEEE Transactions on Vehicular Technology*, 66(11):10283–10295, 2017.

[28] Sam Curry. Web hackers vs. the auto industry: Critical vulnerabilities in ferrari, bmw, rolls royce, porsche, and more, 2023. [Online]. Available: https://samcurry.net/web-hackers-vs-the-auto-industry/. [Accessed 01.18.2024].

[29] Robert I Davis, Alan Burns, Reinder J Bril, and Johan J Lukkien. Controller area network (can) schedulability analysis: Refuted, revisited and revised. *Real-Time Systems*, 35:239–272, 2007.

[30] Heini Bergsson Debes and Thanassis Giannetsos. Zekro: Zero-knowledge proof of integrity conformance. In *Proceedings of the 17th International Conference on Availability, Reliability and Security*, pages 1–10, 2022.

[31] Edlira Dushku, Md. Masoom Rabbani, Jo Vliegen, An Braeken, and Nele Mentens. Prove: Provable remote attestation for public verifiability. *Journal of Information Security and Applications*, 75:103448, 2023.

[32] Nada El Kassem. *Lattice-Based Direct Anonymous Attestation.* PhD thesis, University of Surrey, 2020.

[33] Nada El Kassem, Liqun Chen, Rachid El Bansarkhani, Ali El Kaafarani, Jan Camenisch, Patrick Hough, Paulo Martins, and Leonel Sousa. More efficient, provably-secure direct anonymous attestation from -lattices. *Future Generation Computer Systems*, 99:425–458, 2019.

[34] Robert Escherich, Ingo Ledendecker, Carsten Schmal, Burkhard Kuhls, Christian Grothe, and Frank Scharberth. She: Secure hardware extension-functional specification, version 1.1. *Hersteller Initiative Software (HIS) AK Security*, 2009.

[35] Robert Bosch GmbH. Automotive ip modules - can xl - next step in can evolution, 2023. [Online]. Available: https://www.bosch-semiconductors.com/ip-modules/can-protocols/can-xl/. [Accessed 01.22.2024].

[36] GNU Project. GCC, the GNU Compiler Collection. https://gcc.gnu.org. [Online; accessed 28-August-2024].

[37] S Dov Gordon, Jonathan Katz, and Vinod Vaikuntanathan. A group signature scheme from lattice assumptions. In *Advances in Cryptology-ASIACRYPT 2010: 16th International Conference on the Theory and Application of Cryptology and Information Security*, pages 395–412. Springer, 2010.

[38] Wouter Hellemans, Md Masoom Rabbani, Bart Preneel, and Nele Mentens. Yes we can! towards bringing security to legacy-restricted controller area networks. a review. In *Proceedings of the 20th ACM International Conference on Computing Frontiers*, pages 352–357, 2023.

[39] IBM. IBM's TPM 2.0 TSS. https://sourceforge.net/projects/ibmtpm20tss/. [Online; accessed 28-August-2024].

[40] Ahmad Ibrahim, Ahmad-Reza Sadeghi, and Gene Tsudik. US-AID: Unattended Scalable Attestation of IoT Devices. In *2018 IEEE 37th Symposium on Reliable Distributed Systems SRDS '18*, pages 21–30, 2018.

[41] Infineon. A safe for sensitive data in the car: Volkswagen relies on tpm from infineon, "2019. [Online]". Available: "https://www.infineon.com/cms/en/about-infineon/press/market-news/2019/INFATV201901-030.html". [Accessed 01.18.2024].

[42] Infineon. 32-bit AURIX™ TriCore™ Microcontroller, 2024. [Online]. Available: https://www.infineon.com/cms/en/product/microcontroller/32-bit-tricore-microcontroller/. [Accessed 05.27.2024].

[43] Hyo Jin Jo and Wonsuk Choi. A survey of attacks on controller area networks and corresponding countermeasures. *IEEE Transactions on Intelligent Transportation Systems*, 23(7):6123–6141, 2021.

[44] Robert Michael Kaster. *Automotive Software Attestation: Self, Remote, and Peer-Building Trust in Autonomous Driving Safety Systems*. PhD thesis, University of Michigan-Dearborn, 2024.

[45] Mohammed Khodari, Abhimanyu Rawat, Mikael Asplund, and Andrei Gurtov. Decentralized firmware attestation for in-vehicle networks. In *Proceedings of the 5th on Cyber-Physical System Security Workshop*, pages 47–56, 2019.

[46] Florian Kohnhäuser, Dominik Püllen, and Stefan Katzenbeisser. Ensuring the safe and secure operation of electronic control units in road vehicles. In *2019 IEEE Security and Privacy Workshops (SPW)*, pages 126–131. IEEE, 2019.

[47] F. Kohnhäuser, N. Büscher, and S. Katzenbeisser. A Practical Attestation Protocol for Autonomous Embedded Systems. In *2019 IEEE European Symposium on Security and Privacy (Euro S & P 2019)*, pages 263–278, 2019.

[48] Karl Koscher, Alexei Czeskis, Franziska Roesner, Shwetak Patel, Tadayoshi Kohno, Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, et al. Experimental security analysis of a modern automobile. In *2010 IEEE symposium on security and privacy*, pages 447–462. IEEE, 2010.

[49] Boyu Kuang, Anmin Fu, Shui Yu, Guomin Yang, Mang Su, and Yuqing Zhang. ESDRA: An Efficient and Secure Distributed Remote Attestation Scheme for IoT Swarms. *IEEE Internet of Things Journal*, 2019.

[50] KEEN Security Lab. Experimental security assessment of BMW cars: A summary report, 2018. [Online]. Available: https://keenlab.tencent.com/en/whitepapers/Experimental_\\Security_Assessment_of_BMW_Cars_by_KeenLab.pdf. [Accessed 01.18.2024].

[51] KEEN Security Lab. Experimental security assessment of mercedes-benz cars, 2021. [Online]. Available: https://keenlab.tencent.com/en/2021/05/12/Tencent-Security-Keen-Lab-Experimental-Security-Assessment-on-Mercedes-Benz-Cars/. [Accessed 01.18.2024].

[52] Brooke Lampe and Weizhi Meng. Intrusion detection in the automotive domain: A comprehensive review. *IEEE Communications Surveys & Tutorials*, 25(4):2356–2426, 2023.

[53] Dongxiao Liu, Jianbing Ni, Xiaodong Lin, and Xuemin Shen. Anonymous group message authentication protocol for lte-based v2x communications. *Internet Technology Letters*, 1(2):e25, 2018.

[54] Anna Lysyanskaya, Ronald L Rivest, Amit Sahai, and Stefan Wolf. Pseudonym systems. In *Selected Areas in Cryptography: 6th Annual International Workshop, SAC'99*, pages 184–199. Springer, 2000.

[55] Hafizah Mansor, Konstantinos Markantonakis, Raja Naeem Akram, and Keith Mayes. Don't brick your car: Firmware confidentiality and rollback for vehicles. In *2015 10th International Conference on Availability, Reliability and Security*, pages 139–148. IEEE, 2015.

[56] Mohamad Mansouri, Wafa Ben Jaballah, Melek Önen, Md Masoom Rabbani, and Mauro Conti. Fadia: Fairness-driven collaborative remote attestation. In *Proceedings of the 14th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, WiSec '21, page 60–71, 2021.

[57] Charlie Miller and Chris Valasek. Remote exploitation of an unaltered passenger vehicle. In *Proc. Black Hat USA*, pages 1–91, 2015.

[58] MIRACL. The MIRACL Core Cryptographic Library. https://github.com/miracl/core/tree/master. [Online; accessed 28-August-2024].

[59] Sen Nie, Ling Liu, and Yuefeng Du. Free-fall: Hacking tesla from wireless to CAN bus. pages 1–16, 2017.

[60] NXP. S32 Automotive Processors, 2024. [Online]. Available: https://www.nxp.com/products/processors-and-microcontrollers/arm-processors/s32-automotive-processors:AUTOMOTIVE-MPUS. [Accessed 05.27.2024].

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

[62] OpenSSL Project. Welcome to the OpenSSL Project. https://github.com/openssl/openssl. [Online; accessed 28-August-2024].

[63] Lukas Petzi, Ala Eddine Ben Yahya, Alexandra Dmitrienko, Gene Tsudik, Thomas Prantl, and Samuel Kounev. SCRAPS: Scalable collective remote attestation for Pub-Sub IoT networks with untrusted proxy verifier. In *31st USENIX Security Symposium (USENIX Security 22)*, 2022.

[64] M. M. Rabbani, J. Vliegen, J. Winderickx, M. Conti, and N. Mentens. Shela: Scalable heterogeneous layered attestation. *IEEE Internet of Things Journal*, 6(6):10240–10250, 2019.

[65] Renesas. RH850 Automotive MCUs, 2024. [Online]. Available: https://www.renesas.com/us/en/products/microcontrollers-microprocessors/rh850-automotive-mcus. [Accessed 05.27.2024].

[66] Ronald L Rivest, Adi Shamir, and Yael Tauman. How to leak a secret. In *Advances in Cryptology—ASIACRYPT 2001: 7th International Conference on the Theory and Application of Cryptology and Information Security Gold Coast*, pages 552–565. Springer, 2001.

[67] Reiner Sailer, Xiaolan Zhang, Trent Jaeger, and Leendert Van Doorn. Design and implementation of a tcg-based integrity measurement architecture. In *USENIX Security symposium*, volume 13, pages 223–238, 2004.

[68] Yannick Seurin. On the exact security of schnorr-type signatures in the random oracle model. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 554–571. Springer, 2012.

[69] Rodrigo Vieira Steiner and Emil Lupu. Attestation in wireless sensor networks: A survey. *ACM Comput. Surv.*, 49(3), sep 2016.

[70] STMicroelectronics. SPC5 32-bit Automotive MCUs , 2024. [Online]. Available: https://www.st.com/en/automotive-microcontrollers/spc5-32-bit-automotive-mcus.html. [Accessed 05.27.2024].

[71] Ivan Studnia, Vincent Nicomette, Eric Alata, Yves Deswarte, Mohamed Kaâniche, and Youssef Laarouchi. A survey of security threats and protection mechanisms in embedded automotive networks. In *2013 43rd Annual IEEE/IFIP Conference on Dependable Systems and Networks Workshop (DSN-W)*, pages 1–12. IEEE, 2013.

[72] UNECE. UN regulations on cybersecurity and software updates to pave the way for mass roll out of connected vehicles. Available: https://unece.org/sustainable-development/press/un-regulations-cybersecurity-and-software-updates-pave-way-mass-roll. [Accessed 01.19.2024], 2020. [Online].

[73] Jo Van Bulck, Jan Tobias Mühlberg, and Frank Piessens. Vulcan: Efficient component authentication and software isolation for automotive control networks. In *Proceedings of the 33rd Annual Computer Security Applications Conference*, pages 225–237, 2017.

[74] Xu An Wang, Jianfeng Ma, Fatos Xhafa, Baodong Qin, and Mingwu Zhang. New efficient chosen ciphertext secure elgamal encryption schemes for secure cloud storage service. *International Journal of Web and Grid Services*, 13(3):246–269, 2017.

[75] Stephan Wesemeyer, Christopher JP Newton, Helen Treharne, Liqun Chen, Ralf Sasse, and Jorden Whitefield. Formal analysis and implementation of a tpm 2.0-based direct anonymous attestation scheme. In *Proceedings of the 15th ACM Asia Conference on Computer and Communications Security*, pages 784–798, 2020.

[76] Marko Wolf and Timo Gendrullis. Design, implementation, and evaluation of a vehicular hardware security module. In *Information Security and Cryptology-ICISC 2011: 14th International Conference*, pages 302–318. Springer, 2012.

[77] Yanjiang Yang, Zhuo Wei, Youcheng Zhang, Haibing Lu, Kim-Kwang Raymond Choo, and Haibin Cai. V2x security: A case study of anonymous authentication. *Pervasive and Mobile Computing*, 41:259–269, 2017.

[78] Bidi Ying and Amiya Nayak. Anonymous and lightweight authentication for secure vehicular networks. *IEEE Transactions on Vehicular Technology*, 66(12):10626–10636, 2017.

Appendix

1. Data Availability

Our research strictly adheres to ethical guidelines and does not involve the use of any personal information. We have ensured that there are no vulnerabilities disclosed, and the study does not incorporate any data or information that could potentially lead to harm or losses for individuals or companies. Furthermore, all aspects of our research were conducted entirely in a lab setting, minimizing any risk of unintended consequences or ethical concerns. The source code for our PoC implementation is available: https://gitlab.esat.kuleuven.be/wouter.hellemans/spark.

2. Group Signature

Formally, a group signature scheme

$$GS = (G.KeyGen, G.Sign, G.Vrfy, G.Open)$$

is a collection of four polynomial-time algorithms defined as follows:

- The group key-generation algorithm $G.KeyGen(1^\lambda, 1^N)$ is a randomized algorithm that takes a security parameter 1^λ and the group size 1^N as inputs, and outputs $\Big((X,Y), x_T,\ x_0^{[N]}\Big)$, where (X,Y) corresponds to the issuer public key, x_T is the Tracer's key, and $x_0^{[N]}$ is a list of N signing keys with x_0^i represents the signing key of the i^{th} group member.
- The group signature algorithm $G.Sign(x_0^i, m)$ is a randomized algorithm that takes a signing key x_0^i and a message m as inputs and outputs a signature σ on m.
- The group verification algorithm $G.Vrfy\Big((X,Y), m, \sigma\Big)$ is a deterministic algorithm that takes as input the issuer public key (X,Y), a message m, and a signature σ, and outputs either accept or reject, respectively.
- The opening algorithm $G.Open(x_T, m, \sigma)$ is a deterministic algorithm that takes a tracing key x_T, a message m, and a signature σ as inputs, and outputs the identity $i \in [1, N]$.

We formally define the main security properties of a Group signature:

Definition A.1. (Unforgeability): We define the unforgeability of a Group Signature

$$GS = (G.KeyGen, G.Sign, G.Vrfy, G.Open)\,.$$

In this game, an adversary is required to output a valid forgery σ^* on a chosen message m^* after several signing queries.
System Setup: Keys are generated using $G.KeyGen$. The adversary $\mathcal{A}$ is given the group's public parameters while the private signing keys are never released to $\mathcal{A}$. Note that in our scheme each group member is a branch that consists of an Edge with its connected children IoT devices.
Signature Queries: The adversary issues signing queries on messages $m_1, m_2, \ldots m_l$ for group members $1, \ldots, l$, for some $l \in \mathbb{N}$. To each query m_i where $i \in [1, l]$, a challenger responds by running the $G.Sign$ algorithm to generate a group signature σ_i on m_i with the group member i's signing key, which was generated in the Key Setup Phase and outputs σ_i to the adversary.
Output: At this stage, the adversary outputs a message–signature pair (m^*, σ^*). The experiment outcome can be analyzed as follows: The adversary wins if

- σ^* is a valid group signature of m^* according to Verify algorithm $G.Vrfy$;
- (m^*, σ^*) is not among the pairs (m_i, σ_i) that were queried before.

We define the advantage of an adversary $\mathcal{A}$ in attacking the unforgeability of our group signature scheme as the probability that $\mathcal{A}$ wins the above game.

Definition A.2. (Anonymity): A group signature scheme

$$GS = (G.KeyGen, G.Sign, G.Vrfy, G.Open)$$

is anonymous if for all probabilistic polynomial-time adversaries $\mathcal{A}$, the advantage of $\mathcal{A}$ in the following experiment is negligible in λ:

- Compute $((X,Y), x_T, x_0^{[N]}) \leftarrow G.KeyGen(1^\lambda, 1^N)$ and give $((X,Y), x_0^{[N]})$ to $\mathcal{A}$.
- $\mathcal{A}$ outputs distinct identities $i_0, i_1 \in [1, N]$, along with a message m. A random bit b is chosen, and $\mathcal{A}$ is given $G.Sign(x_{0_{i_b}}, m)$. Finally, $\mathcal{A}$ outputs a bit b'.
- $\mathcal{A}$ succeeds if $b' = b$ we denote this case by "Succ". The advantage of $\mathcal{A}$ is Pr[Succ] - $\frac{1}{2}$.

Definition A.3. (Traceability): A group signature scheme

$$GS = (G.KeyGen, G.Sign, G.Vrfy, G.Open)$$

is traceable if for all probabilistic polynomial-time adversaries $\mathcal{A}$, the success probability of $\mathcal{A}$ in the following experiment is negligible in n:

- Compute $\Big((X,Y), x_T, x_0^{[N]}\Big) \leftarrow G.KeyGen(1^\lambda, 1^N)$ and give (X,Y) and x_T to $\mathcal{A}$.
- $\mathcal{A}$ may query the following oracles:
  - A Corrupt oracle that on input $i \in [N]$ returns x_0^i.
  - A Signing oracle that on input i, m outputs a signature $\sigma \leftarrow G.Sign(x_0^i, m)$.
- $\mathcal{A}$ outputs a message m and a signature σ.
  Let $\mathcal{C}$ be the set of corrupt identities. $\mathcal{A}$ succeeds if

  $$G.Vrfy\Big((X,Y), m, \sigma\Big) = 1$$

  and σ was never queried by any member that doesn't belong to $\mathcal{C}$, however $G.Open(x_T, m, \sigma) \notin \mathcal{C}$.

3. Network Overhead Analysis

In this appendix, we derive the overhead of our protocol on the network resources. In our analysis, we assume that SPARK leverages the BN_P256 curve, adopted by the TCG. As such, every point on $\mathbb{E}$ can be represented with maximum 512 bits. Furthermore, the Zonal Gateway and every ECU have to exchange values in $\mathbb{Z}_q$, which we consider to consist of maximum 256 bits. In the following, we provide an analysis of the overhead introduced by these

messages for (1) CAN, (2) CAN FD, and (3) CAN XL. The parameters of these three protocols are summarized in Table 4.

TABLE 4: Summary of maximum transmission speed and payload size in different CAN specifications

	CAN	CAN FD	CAN XL
$R_{arbitration,max}$ (Mbit/s)	1*	1	1
$R_{data,max}$ (Mbit/s)	1*	8	20
Maximum payload (bytes)	8	64	2048

* CAN does not support different data rates during the arbitration and data phase.

Since packets in CAN-based protocols are subject to bit-stuffing, we consider the worst-case transmission time (i.e., maximum bit stuffing) in our network overhead analysis. Furthermore, to allow for a fair comparison between the different CAN standards, we use the base format with 11-bit identifiers[2]. The worst-case transmission time for the different CAN standards can be found as:

1) CAN [29]:

$$t_m = (55 + 10p)\tau_{bit}$$

2) CAN FD [11]:

$$t_m = 32\tau_{arb} + (28 + 5\lceil\frac{p-16}{64}\rceil + 10p)\tau_{data}$$

3) CAN XL [10]:

$$t_m = 37\tau_{arb} + (119 + \lfloor\frac{109+8p}{10}\rfloor + 8p)\tau_{data}$$

with, p the number of bytes in the payload, τ_{bit} the bit time, τ_{arb} the bit time during the arbitration phase, and τ_{data} the bit time during the data transmission phase.

In Table 5, we present an analysis of the worst-case overhead on the network for a single ECU, assuming the maximum transmission speeds of the CAN standards. Additionally, we analyze the network overhead in terms of the number of ECUs in the network in Figure 10. Our results demonstrate that for the considered branch size of 32 ECUs, the overhead on the network is limited to 120.960 ms, 14.740 ms, and 8.7184 ms for CAN, CAN FD, and CAN XL, respectively. This corresponds to bus load increments of 12.09%, 1.47%, and 0.871%, respectively. Furthermore, it can be found that SPARK significantly benefits from the higher transmission rate and longer payload offered by CAN FD compared to CAN. We note that the additional improvement offered by CAN XL w.r.t. CAN FD is less significant. This is because the payloads in the different messages are relatively small and CAN XL suffers from a longer arbitration phase. Finally, it is important to note that the increase in bandwidth only applies during the attestation process. As such, SPARK does not compromise the normal operation of the vehicle.

2. In contrast to the CAN and CAN FD standards, CAN XL does not have support for extended identifiers.

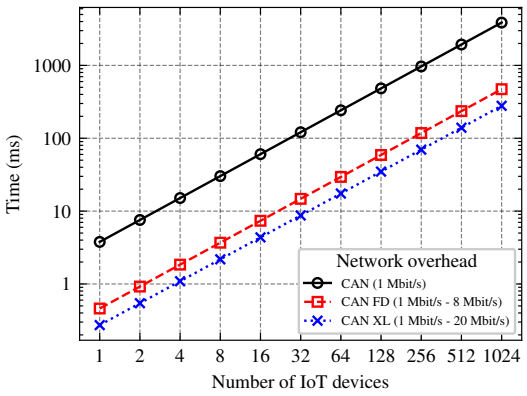

Figure 10: Network-level overhead for communication between ECU and Zonal Gateway during the `Attestation` phase. The X- and Y-axes are of a logarithmic nature.

4. Protocol Timings

In this appendix, we summarize the scalability of SPARK during the attestation and verification for branch sizes of 1 to 1024 ECUs. Moreover, we compare the timings with and without tracing activated. The results are presented in Table 6.

TABLE 5: Summary of the per-ECU network-level overhead during the `Attestation` phase for different CAN specifications.

	CAN			CAN FD			CAN XL		
	# messages	payload	time (μs)	# messages	payload	time (μs)	# messages	payload	time (μs)
E'_k	8	8	1080	1	64	116.125	1	64	71.65
$\mathcal{L}_k$	4	8	540	1	32	76.125	1*	96*	85.70*
R_k	8	8	1080	1	64	116.125			
c	4	8	540	1	32	76.125	1	32	57.55
s_k	4	8	540	1	32	76.125	1	32	57.55
`Total`	3780 μs/IoT			460.625 μs/IoT			272.45 μs/IoT		

* Please note that due to the larger payload of CAN XL, the exchange of $\mathcal{L}_k$ and R_k can be achieved with a single message

TABLE 6: Overview of the device-level overhead in SPARK: (1) on the Zonal Gateway during the `Attestation` with and without tracing, (2) total overhead during the attestation, and (3) on the Central Gateway during the `Verification` with and without tracing. The scalability is evaluated for network branch sizes ranging from 1 to 1024

# IoT	Edge (ms)	Edge (Tracing) (ms)	Total M4 (ms)	Total M0+ (ms)	Total ESP32 (ms)	Verification (ms)	Verification (Tracing) (ms)
1	148.390	284.384	294.623	287.334	182.800	40.570	51.203
2	150.302	285.991	296.536	289.246	184.712	43.894	54.322
4	153.221	289.595	299.455	292.165	187.632	52.475	62.577
8	158.575	294.521	304.809	297.519	192.985	67.624	76.903
16	170.432	305.969	316.666	309.376	204.842	100.568	110.576
32	193.805	328.848	340.039	332.749	228.215	166.080	174.240
64	236.662	374.509	382.896	375.606	271.073	297.410	305.781
128	327.234	461.950	473.468	466.178	361.645	556.022	567.205
256	504.575	640.421	650.809	643.519	538.985	1071.986	1084.521
512	861.046	997.743	1007.280	999.990	895.456	2106.163	2119.283
1024	1575.428	1712.185	1721.662	1714.372	1609.839	4173.503	4185.389