Grant Agreement No.: 952697
Call: H2020-SU-ICT-2018-2020
Topic: SU-ICT-02-2020
Type of action: RIA

D3.7 ASSURED SECURE AND SCALABLE AGGREGATE NETWORK ATTESTATION – VERSION 2

Revision: v.1.0

Work package	WP 3
Task	Task 3.5
Deliverable lead	TUDA
Version	1.00
Editors	Richard Mitev (TUDA), Philip Rieger (TUDA), Nada El Kassem (SURREY)
Reviewers	Liquin Chen (SURREY), Edlira Dushku (DTU)
Abstract	Deliverable D3.7 provides the design and implementation details on the final version of the ASSURED Swarm Attestation scheme, which was initially presented in D3.6. The purpose of the scheme is to enable the simultaneous attestation of multiple devices (leveraging the single-Prover CFA or CIV enablers, as presented in D3.3), in a manner that is more computationally efficient than attesting to the correctness of each device individually. The solution provided by ASSURED offers a wide variety of features, such as the provision of **privacy-preserving attestation**, including both **identity privacy** (i.e., the protection of the identity of the Prover devices), and **attestation evidence privacy** (to ensure that the Prover devices divulge no information about their configuration state, which could be used by a malicious party to carry out implementation disclosure attacks). In addition, the designed scheme provides traceability and linkability features, which enable the identification of a compromised device in case of a failed attestation, as well as the revocation of that device, based on the use of pseudonyms. An additional important update presented in this deliverable entails the consideration of dynamic network topologies, which considers the mobility of the swarm devices. Finally, experimental results are provided in order to evaluate the performance of the designed scheme, as well as a security analysis of the protocol. To the best of our knowledge, this is the first instance of such a scheme that can reconcile swarm attestation and safety-critical operations in simple IoT devices.
Keywords	Trusted Computing, Swarm Attestation, Attestation Evidence Privacy, Identity Privacy, Dynamic Network Topology

Funded by the Horizon 2020 Framework Programme of the European Union

Document Revision History

Version	Date	Description of change	List of contributors
v0.1	18.01.2023	Table of Contents provided, summarizing the components and functionalities of ASSURED to be covered by the deliverable	Richard Mitev (TUDA), Philip Rieger (TUDA)
v0.15	17.02.2023	Definition of the system model, security and privacy requirements, and building blocks of the Swarm Attestation scheme. Details on trust assumptions and trust modelling to be used as part of security proof. (Chapter 3)	Edlira Dushku, Heini Bergsson Debes, Benjamin Larsen, Nicola Dragoni (DTU) Richard Mitev, Philip Rieger, Marco Chilese, David Koisser (TUDA) Nada El Kassem (SURREY) Thanassis Giannetsos, Dimitris Karras, Stefanos Vasileiadis, Alexandros Sampanis (UBITECH) Ilias Aliferis (UNIS)
v0.2	03.03.2023	Details on the considered dynamic network topology and threat model to be addressed by the Swarm Attestation scheme. (Chapter 3)	Thanassis Giannetsos, Dimitris Karras (UBITECH) Edlira Dushku, Heini Bergsson Debes, Benjamin Larsen, Nicola Dragoni (DTU) Liqun Chen, Nada El Kassem (SURREY)
v0.3	19.03.2023	Details on the building blocks of the Swarm Attestation scheme, namely aggregated signatures, DAA, ring signatures, and property-based attestation. (Chapter 3)	Thanassis Giannetsos, Dimitris Karras (UBITECH) Edlira Dushku, Heini Bergsson Debes, Benjamin Larsen, Nicola Dragoni (DTU) Liqun Chen, Nada El Kassem (SURREY)
v0.4	31.03.2023	Description and formalization of the Setup phase of the Swarm Attestation scheme, including Edge and IoT device enrollment. (Chapter 4)	Edlira Dushku, Heini Bergsson Debes, Benjamin Larsen, Nicola Dragoni (DTU) Nada El Kassem (SURREY)
v0.45	13.04.2023	Description and formalization of the Attestation and Verification phase of the Swarm Attestation scheme. (Chapter 4)	Edlira Dushku, Heini Bergsson Debes, Benjamin Larsen, Nicola Dragoni (DTU) Nada El Kassem (SURREY)
v0.5	18.04.2023	Initial review of the Swarm Attestation protocol, including building blocks and algorithms.	Liquin Chen (SURREY), Thanassis Giannetsos (UBITECH)
v0.55	27.04.2023	Clarifications to the descriptions of the building blocks, updates on the algorithms in order to clarify attestation and verification process, and updates on the revocation process.	Thanassis Giannetsos, Dimitris Karras (UBITECH) Edlira Dushku, Heini Bergsson Debes, Benjamin Larsen, Nicola Dragoni (DTU)
v0.6	10.05.2023	Security analysis of the Swarm Attestation scheme, including security proof and protection against the considered threat model. (Chapter 4)	Nada El Kassem, Liqun Chen (SURREY)
v0.7	17.05.2023	Implementation details, experimental results, and performance evaluation of the Swarm Attestation scheme. (Chapter 5)	Thanassis Giannetsos, Dimitris Karras, Stefanos Vasileiadis, Alexandros Sampanis (UBITECH) Ahmad Atali, Meni Onrebach (MLNX)
v0.8	24.05.2023	Summary of the features and updates based on final version of the ASSURED Swarm Attestation scheme, including high-level overview and building blocks. (Chapter 2)	Thanassis Giannetsos, Dimitris Karras (UBITECH) Edlira Dushku, Heini Bergsson Debes, Benjamin Larsen, Nicola Dragoni (DTU)
v0.9	26.05.2023	Finalization of the introduction and conclusions section of the deliverable (Chapters 1 and 6)	Thanassis Giannetsos, Dimitris Karras (UBITECH)
v0.95	02.06.2023	Review of the entire deliverable	Liquin Chen (SURREY), Thanassis Giannetsos (UBITECH)
v1.0	16.06.2023	Update of the deliverable based on the review comments	Thanassis Giannetsos, Dimitris Karras (UBITECH)

Funded by the Horizon 2020 Framework Programme of the European Union

Editors

Richard Mitev (TUDA), Philip Rieger (TUDA), Nada El Kssem (SURREY)

Contributors (ordered according to beneficiary numbers)

Edlira Dushku, Heini Bergsson Debes, Benjamin Larsen (DTU)

Richard Mitev, Philip Rieger, Marco Chilese, David Koisser (TUDA)

Liqun Chen, Nada El Kassem (SURREY)

Ahmad Atali, Meni Onrebach (MLNX/NVIDIA)

Thanassis Giannetsos, Dimitris Karras, Stefanos Vasileiadis, Alexandros Sampanis (UBITECH)

Ilias Aliferis (UNIS)

Funded by the Horizon 2020 Framework Programme of the European Union

DISCLAIMER

COPYRIGHT NOTICE

Project co-funded by the European Commission in the H2020 Programme		
Nature of the deliverable:	**R**	
Dissemination Level		
PU	Public, fully open, e.g. web	✓
CL	Classified, information as referred to in Commission Decision 2001/844/EC	
CO	Confidential to ASSURED project and Commission Services	

* R: Document, report (excluding the periodic and final reports)

Funded by the Horizon 2020 Framework Programme of the European Union

Executive Summary

The number and variety of special-purpose computing devices is increasing drastically, in multiple (safety-critical) application domains, that leverage such cyber-physical systems in “smart” settings; such as smart factories, critical infrastructures, automotive (e.g., Cooperative, Connected and Automated Mobility (CCAM) [34,42]). As society becomes increasingly accustomed to being surrounded by, and dependent on, such devices, their security and safety becomes extremely important. Furthermore, since such systems usually support **real-time applications with strict timing requirements [13], integrating new security and assurance controls is a challenging task that needs to be handled gracefully so that such a consolidation does not influence the device’s runtime behavior**.

Compounding this issue, ASSURED introduced a set of novel **attestation enablers** [24] (implemented as a **SW/HW co-design**) allowing both the **remote and local verification of the configuration and behavioral correctness of a device**. These controls targeted the software and hardware (firmware) layers and covering all phases of a device’s execution; from the **trusted boot and integrity measurement** of a CPS, enabling the generation of static, boot-time or load-time evidence of the system’s components correct configuration **(Configuration Integrity Verification (CIV) [30,43])**, to the **runtime behavioral attestation** of those safety-critical components of a system providing strong guarantees on the correctness of the **control- and information-flow properties** (**Control-Flow Attestation** [50]), thus, enhancing the performance and scalability when composing secure systems from potentially insecure components. These security services were also enriched with **zero-knowledge capabilities** [27–29] for been able to allow a Verifier (Vrf) device to establish a dynamic root-of-trust in the Prover (Prv) without the need of disclosing the actual state of the Prv, thus, weakening the overall trust assumptions that most of the existing attestation schemes suffer from that hinders their applicability to services with inherent zero trust guarantees for all participating actors and stakeholders. This also provided the necessary means for the design of an **Enhanced Direct Anonymous Attestation (DAA)** [44] protocol towards the provision of privacy-preserving platform authentication capabilities while also enabling device-controlled anonymity and unlinkability when exchanging safety-critical data with other actors in the overall service graph chain. Finally, an additional **Jury-based attestation mechanism** was also provided for allowing the handling of disagreements (during the execution of an attestation process) between the Prv and the Vrf: In case of a dispute on the verifiability of evidence, provided by the Prv, other constituent devices will be triggered to help by providing an additional layer of verification on the correctness/honesty of the initial Prv and Vrf devices so that they can identify the culprit (dishonest) node.

While all these attestation primitives achieve high efficiency (as detailed in Deliverable D6.3 [23] through an extensive set of experiments), showcasing that this type of SW/HW co-design approach followed is particularly promising for low-end embedded devices, they target single-Prover and single-Verifier attestation tasks. This is the first step, in the overarching vision of ASSURED, towards the creation of “*communities of trust*” where (dynamic) trust relationships can be assessed and established between entities, starting from bi-lateral interactions between two single system components and continuing as such systems get connected to ever larger entities. But *how can we transfer such security claims on the correctness of the properties of single systems to hierarchical composition of systems (“Systems-of-Systems”)?*. In this context, ASSURED has proposed a new variant of **Swarm Attestation** (introduced in Deliverable D3.6 [22]) that enables a **root Verifier to check the sanity of a set (swarm) of devices in a privacy-preserving manner**; concealing the identities of the Prv devices, and only in the case of a possible compromise

detection does the scheme allows for tracing back a failed attestation to the swarm device that caused the failure (instead of isolating the entire swarm).

The ASSURED Swarm Attestation (SA) scheme aims to fulfill the security and privacy requirements set forth by the Next-Generation Smart Connectivity SoSes towards the evolution of such safety-critical SoS from local, stand-alone systems into safe and secure solutions distributed over the continuum from cyber-physical end devices, to edge servers and cloud facilities. Specifically, we first established the required **security and trustworthiness properties** that need to be achieved, as well as the novel provided feature of Prv **identity privacy** through the integration of the ASSURED **Enhanced Traceable Direct Anonymous Attestation (DAA)** scheme. This enables us to both **protect the system from malicious parties**, and to **protect devices from untrusted verifying entities**. As aforementioned, the purpose of this scheme is to enable unlinkable attestation, so that no successful attestation result can be linked back to the source device, and no potentially system identifiable information can be leaked.

However, what is still lacking is the safeguarding of **attestation evidence privacy** so as to be able to avoid implementation disclosure attacks. As was also described in D3.6 [22], existing swarm attestation schemes lack of privacy, during the execution of all underlying attestation tasks (e.g., CIV, CFA, etc.) as Prover devices must comprehensively disclose program execution or configuration details, which is unattractive when such information should remain a secret. **Enabling Verifiers to have full knowledge of a Prover's memory layout and configuration profile (e.g., type of OS loaded) renders them prone to privacy breaches and implementation disclosure attacks:** "honest-but-curious" verifying entities can exfiltrate sensitive information on the device's configuration, safety-critical functionalities, etc., which can be used to harm its general availability. In addition, exposing all execution details for verification contradicts the **Zero Trust** principle (assumed in ASSURED [19]) that dictates that **no initial trust can be assumed between entities**. Revealing information on running processes of a system can enable adversaries to benefit from such knowledge towards mounting run-time attacks against the program's codebase. Compounding this issue, what is needed, is the ability to be able to verify the correct execution and/or configuration profile of a device (in the swarm) without, however, been able to pinpoint what is the actual device profile that is running.

This is the focus of this deliverable: **To present the second and final version of the ASSURED Swarm Attestation capabilities supporting both identity and attestation evidence privacy**. To achieve this milestone, we have enhanced the previous SA variant with the use of **ring signatures** (to work in tandem with the crypto primitives of Enhanced DAA and aggregate signatures already employed) so that the root Verifier can attest to the correctness of the entire swarm not only without the need of each device disclosing any details on their running state but also considering the difference in the configuration profile of each comprising device. In other words, the root Vrf can ascertain that each device (of the target swarm) is at a correct state, as depicted by a set of (hash) "*golden states*" characterized as trusted by the application owner, but without knowing the identity of each device nor the state that is associated to each device. This is achieved by protecting the identity of the signing IoT device with hiding its public key, through the use of anonymized DAA credentials. At the same time, the properties of controlled-linkability and accountability (resulting into the credential revocation of failed-to-attest devices) are still accounted for in this new SA variant.

In order to verify the correctness and security of the ASSURED SA scheme, we also provide a rigorous mathematical security analysis and proof, based on the use of the **Universal Composability (UC)** model. This guarantees that the security of the SA protocol is preserved under an internal composition operator, but also provides stronger guarantees for maintaining the security

of the scheme in any context, even in the presence of an unbounded number of attackers and independent of the capabilities of the attacker. The UC framework achieves the formal verification of the security proofs under these strong assumptions, thus serving towards the overall vision of ASSURED for scalable and lightweight crypto protocols that can be abstracted and transferred to different types of application domains with varying security, privacy, and trust requirements.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The exponential proliferation of low-cost embedded devices and Internet-of-Things (IoT) gadgets has increased their usage in Next-Generation Smart Connectivity "Systems-of-Systems" (SoS) [44], motivated by the need to **cooperatively execute safety-critical functions**. However, this approach also turns the devices into attractive cyber-attack targets. Therefore, in the face of an increasing attack landscape, it is **imperative to ensure their correct and safe operation because, by their very nature, these software-based components may not always be in trusted custody**.

Towards this direction and to ensure the correctness of the operational state of a device, *Remote attestation* (RA) [4, 32] has been proposed in order to detect unexpected modifications in the configuration of loaded binaries and check software integrity. However, **most proposed attestation schemes have assumed the existence of a single Prover and a single Verifier, which introduces efficiency and scalability issues in the context of large-scale systems comprising multiple Edge and IoT devices**. In this context, swarm attestation [5] has been proposed, which enables the root Verifier ($\mathcal{V}$) to simultaneously check the sanity of a set (swarm) of devices.

Several swarm attestation schemes have been proposed in the literature. Most are based on the use of a *spanning tree* [5, 14] to aggregate the attestation results, where swarm devices are attested by neighboring devices based on a tree format with the Verifier as a root. The *broadcasting* aggregate pattern addresses this issue, by having each device broadcast its response to the attestation challenge to its neighbors [41]. However, **there has been very limited focus on privacy issues of the devices [3] both as t pertains to identity privacy but also attestation evidence privacy**. Two schemes are proposed in [14], one of which keeps a list of the failed devices and the other does not. In [37], compromised devices and services are identified, and in [32], it is detected whether a device has been infected by a compromised device.

One big challenge, in this context, is the need for a "*trusted*" Verifier to conduct the attestation process and assert the integrity of all swarm IoT devices. This contradicts the **emerging Zero Trust principle with the need of "Never Trust, Always Verify" [49]** and complicates scalability and efficiency due to Verifier complexity. **Such an obsolete assumption renders existing swarm attestation schemes prone to privacy breaches and implementation disclosure attacks under *honest-but-curious* adversaries**. Consider, for instance, the case of automatic collision avoidance of a vehicle where the brake Electronic Control Unit (ECU) and radar sensors are potential targets that need to be checked for their proper functionality in order to assert that the system will work as intended. In the case of a third-party Verifier running at the Multi-Access Edge Computing Layer (MEC), as envisioned by ETSI [49] (more details about the MEC can be found in Chapter 3), **sharing all configuration and internal execution details of such safety-critical components do not only raise significant privacy concerns on the identity of the**

vehicle driver (by linking specific ECU IDs to the overall vehicle profile) but is also not allowed due to contractual constraints.

This sets the challenge ahead: *Can we identify a secure and efficient attestation scheme that can correctly make valid statements about the integrity of both single devices and a swarm of devices in a privacy-preserving but accountable manner?* This means that the designed scheme should be able to provide **verifiable evidence on the correctness of a swarm by concealing the identities of the devices and the attestation evidence based on which the verification process is to be excuted**, and only in the case of a possible compromise detection should it allow for tracing back a failed attestation to the swarm device that caused the failure (instead of isolating the entire swarm).

This is the focus of this Deliverable: Building on top of the first variant of a novel Swarm Attestation protocol that was presented in D3.6 [22], achieving **identity privacy** through the use of our Enhanced Direct Anonymous Attestation (DAA) protocol, we now proceed with the necessary enhancements for also enabling **attestation evidence privacy** through the use of **ring signatures**. *ASSURED is the first of its kind to merge the use of such advanced crypto primitives in a way that results in an efficient protocol capable of been instantiated in resource-constrained embedded devices*. The endmost goal is not to simply reduce the communication overhead compared to the naive approach of applying many time single-Prover attestation tasks, but also enable an additional layer of **attestation intelligence and governance** that can allow Verifier devices to cope with the intrinsic characteristics of the service graph topologies that usually pervade these types of "Systems-of-Systems": (i) Such systems usually demonstrate a **high level of mobility** resulting in continuous changes occuring to the construction of the routing tree since they will need to establish new "*parent relationships*" with other nodes for relaying its communicated data, and (ii) The need to establish **federated trust between the devices in the swarm** so as to manage the *hierarchical topology* that usually characterize such systems where the device at each layer can act as the Verifier for its children or a *tree-based* structure where only the root node can act the root Verifier.

This type of features is one of the motivating factors for the enhanced version of the ASSURED Swarm Attestation mechanism that can support such **dynamic network topologies**, requiring **efficient and continuous authentication capabilities** (between the IoT device and the Edge device acting as the parent node in this particular time frame), and **flexible key management for enabling the verification of (signed) attestation reports in a hierarchical topology**: From intermediate nodes (acting as Verifiers) that might not have established a trust relationship with the Prover and need to query another trusted entity for the correctness of the presented crypto material or offload the verification process (thus, enabling a *verification-as-a-service* model).

1.1 Scope and Purpose

The main purpose of this deliverable is to present the final release of a secure and scalable aggregate network attestation mechanism for managing the trusted activities envisioned within the "Systems-of-Systems" environment. This document describes the Swarm Attestation scheme, which is a key component of the ASSURED attestation framework. Swarm Attestation will enable multiple parties to collectively attest their trustworthiness of their configuration state or the correct execution of a software process to a Verifier. This document expands on the results presented in D3.6, where we described the first version of the Swarm Attestation scheme, and how the attestation task is distributed to all devices in the network. Focus will be placed on the privacy

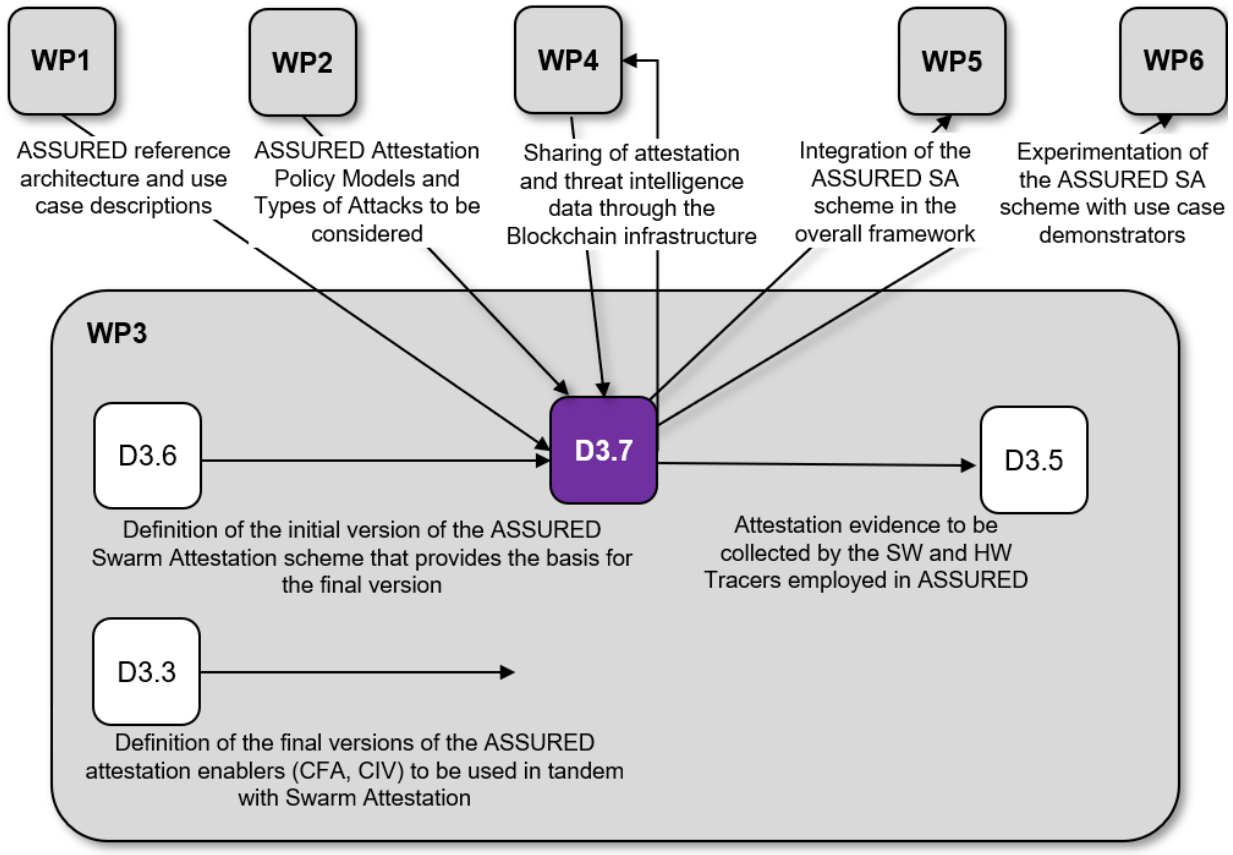

Figure 1.1: Relation of D3.7 with other WPs and Deliverables

concerns of the attestation scheme, and particularly attestation evidence privacy. Attestation usually requires the attested devices (Provers) to reveal all details regarding the program execution and system configuration, making the verifying entity a single point of failure that may enable an attacker to gain sensitive information about all devices that are part of the SoS. In comparison, the decentralized swarm attestation scheme avoids such a single point of failure. We address the challenges of a decentralised attestation scheme by designing attestation primitives that use efficient and lightweight cryptographic operations, such as group-based signatures and rign signatures. This enables the ASSURED framework to effectively verify the trustworthiness even of large networks, consisting of many devices.

1.2 Relation to other WPs and Deliverables

Figure 1.1 depicts the relationship of this deliverable with other Work Packages (WPs), as well as the other tasks within the same WP(3). As aforementioned, the focus of D3.7 is the finalization of the implementation of the ASSURED Swarm Attestation scheme, which provides the capability to attest to the correctness of the operational state of multiple devices simultaneously in a privacy-preserving manner, in tandem with the Control Flow Attestation (CFA) and Configuration Integrity Verification (CIV) schemes, whose final versions are provided in D3.3 [24]. The scheme presented in this delierable builds upon the initial version of the Swarm Attestation scheme, which was presented in D3.6 [22]. In addition, the construction of these attestation schemes dictates the type of attestation evidence to be collected by the SW-based and HW-based tracing schemes presented in D3.5 [25].

The descriptions of the envisioned ASSURED use cases, as well as the security and privacy requirements put forth in this context that need to be fulfilled by the ASSURED SA scheme, were presented in the context of WP1. However, in D3.7, these scenarios are further formulated to highlight those specific needs to be captured by the newly integrated features of the final SA variant and set the scene for the detailed evaluation and benchmarking performed in Chapter 6. WP2 provided the policy models and information on the types of attacks considered, that need to be addressed by the ASSURED attestation toolkit. The results of the ASSURED SA scheme (as well as the other ASSURED attestation enablers) are stored in the Blockchain Infrastructure, which has been analyzed in detail throughout WP4. The integration of the ASSURED SA scheme with the overall framework will be outlined in WP5, and WP6 entaile experimentation activities in the context of the envisioned use cases.

1.3 Deliverable Structure

This deliverable is structured as follows: In **Chapter 2**, we provide a summary of the updates and functionalities of the ASSURED SA scheme presented in this deliverable, as well as a high-level overview of the building blocks and workflow of the scheme. In **Chapter 3** we proceed with a detailed description of the assumed system model especially considering the migration to **dynamic network topologies** better emulating the current needs of complex service graph chains. This is also accompanied with a summary of the attacker's capabilities assumed in ASSURED as well as the list of security and privacy requirements to be achieved. These set the scene against which the formal verification and mathematical proof of the newly designed protocol is based upon. **Chapter 4** constitutes the heart of this deliverable where we provide a detailed analysis of the proposed scheme, including a description of underlying cryptographic primitives employed. A detailed formal security analysis is provided in **Chapter 5**, where it is demonstrated that the designed scheme fulfills the considered security and privacy requirements. Experimental results and evaluations of the offline and online phases of the proposed scheme are provide in **Chapter 6**. Finally, **Chapter 7** concludes the deliverable.

Chapter 2

Additional Features Complementing the ASSURED Swarm Attestation Scheme

The purpose of the ASSURED framework is to provide operational assurance to **large-scale Systems-of-Systems (SoS)**, in order to achieve the **required level of trustworthiness of the entire supply chain ecosystem**. In order to achieve this and to fulfill the **security and privacy requirements** set forth by such services, we have implemented a set of novel attestation enablers, which have been analyzed in detail in D3.2 [18] and D3.3 [24], and are enforced through **dynamically adaptable policies** as mitigation actions to be executed in each Edge device.

The ASSURED **Configuration Integrity Verification (CIV)** [28, 43] and **Control Flow Attestation (CFA)** enablers are able to attest to the correctness of the configuration and behavioural profile of a device, respectively. However, the issue that arises in this type of large-scale ecosystems considered in ASSURED is that, as single-Prover and single-Verifier attestation schemes, there are cases where the attestation of a large number of edge devices separately to be performed by a single Verifier is not efficient, as it requires a large amount of time and computational resources. To this end, we have developed the **ASSURED Swarm Attestation (SA) enabler**, which aims to attest to the correctness of a set (swarm) of devices, in a manner that is **more computationally efficient that attesting each device individually**, but also **attending to the security and privacy requirements the attested devices are subject to**. The first version of the ASSURED SA scheme was presented in D3.6 [22], where we provided information regarding the considered system and adversarial model, network topology, and a detailed description of the designed attestation protocol.

However, it is important to note that there are two core aspects that need to be considered with regards to privacy considerations in the ASSURED SA scheme, namely **identity privacy** and **attestation evidence privacy**. The former refers to the ability of each edge device participating in an attestation process to hide its identity and protect its **unlinkability** and **anonymity**, by not requiring the device to divulge any personally identifiable information. Consider, for instance, the case of the "*Smart Cities*" scenario where a variety of devices (such as cameras and smoke sensors) collect data towards achieving public safety. In this case, we aim to simultaneously attest to the correctness of the operational state of multiple such devices simultaneously while protecting their **identity privacy**, since the collected data may be personally identifiable and may contain privacy-sensitive information. This means that we need to ensure that no such identifiable information will be leaked during the execution of the swarm attestation protocol unless an authorized entity makes a request: In the case (for instance) of police enforcement bodies, it should be possible to have access to the necessary link tokens for associating an (attestation) result back to the data source - especially if we are referring to deployed video cameras where correctness of con-

figuration software profiles need to be attached (as security claims) to any video stream provided and for which such authenticated users should be able to have full linkability and identification of the respective data sources. Identity privacy has been considered in the first version of the SA enabler presented in D3.6 [22].

However, during the attestation process, system traces are collected by the Runtime Tracer to be used as evidence in the attestation process. In the case of CIV, this refers to information on the device configuration, while in the case of CFA, this refers to control-flow graphs extracted during the execution of a software process. If these traces are not safeguarded, it is possible for a malicious party to extract valuable information regarding the device's configuration software profile. The latter pertains to the *lack of privacy*, during the execution of all underlying attestation tasks (i.e., Control-Flow Attestation and/or Configuration Integrity Verification), as Provers must comprehensively disclose program execution or configuration details, which is unattractive when such information should remain secret. **Enabling Verifiers to have full knowledge of a Prover's memory layout and configuration profile (e.g., type of OS loaded) renders them prone to privacy breaches and implementation disclosure attacks:** "honest-but-curious" verifying entities can exfiltrate sensitive information on the device's configuration, safety-critical functionalities, etc., which can be used to harm its general availability. In addition, exposing all execution details for verification contradicts the **Zero Trust** principle (assumed in ASSURED [19]) that dictates that **no initial trust can be assumed between entities**. Revealing information on running processes of a system can enable adversaries to benefit from such knowledge towards mounting run-time attacks against the program's codebase.

Compounding this issue, what is needed, is the ability to be able to verify the correct execution and/or configuration profile of a device (in the swarm) without, however, been able to pinpoint what is the actual device profile that is running. In other words, consider that each (swarm) Prover device produces (and signs) a message m as the result of the attestation process to be sent to the Verifier: **No verifying entity should be able to match the content of m back to the device origin while at the same time the Verifier should be provided with verifiable evidence on the operational assurance of the device in a zero-knowledge manner**.

This feature had not been implemented in the first version of the SA scheme, but is provided in the second version, which is presented in detail in this deliverable based on the use of ring signatures. In the following Table 2.1, we summarize the features of the final version of the Dynamic Swarm Attestation designed in the context of ASSURED. To the best of our knowledge, this is the first instance of such a scheme that can reconcile swarm attestation and safety-critical operation on simple IoT devices.

Feature	Description
Swarm Attestation Efficiency & Scalability	ASSURED can attest a large number of devices in an efficient, effective and privacy-preserving manner, independently of the type of attestation task that needs to be employed. This requirement is about the operation of the swarm attestation scheme that should be **agnostic** to the type of attestation enabler employed: either Control-flow Attestation, Configuration Integrity Verification or Direct Anonymous Attestation. Irrespective, of the types of system properties to be traced (attestation evidence including control-flow graphs, or device configuration), the ASSURED SA is able to operate in a secure and efficient way - even in the case where different devices, in the same swarm, require the attestation of different types of properties, thus, executing different types of attestation tasks.
Heterogeneity	ASSURED considers an IoT swarm that consists of heterogenous devices ranging from low-end IoT device to more powerful edge devices.

Trusted Computing Base	ASSURED leverages the presence of an underlying Trusted Component (TC) for supporting all the necessary capabilities required for the efficient and privacy-preserving attestation of a swarm of devices. This includes *Root-of-Trust for Storage, for Measurement and for Reporting*. This means that the Root-of-Trust for the verification is enacted by the TC of the Prover which adds one extra layer of complexity during the (privacy-preserving) authentication and enrollment of the device into the system: Authenticating the correctness of a device prior to initiating the construction of the necessary keys (to support the entire lifecylce of the device binded to the appropriate key restriction usage policies) is not sufficient. This process needs to also verify that the device is equipped with a valid Trusted Component whose certificate has not been revoked for any reason. ASSURED overcomes this challenge this challenge - identified as "Key Binding problem" - considering all intricacies with the main one been not only on the verification that a (host) device is equipped with a valid TC but also on on the authentication between the TC and the host itself and elaborates on the steps taken in the ASSURED system model for resolving it.
Dynamic Network Topology	ASSURED performs swarm attestation of a network that follows the fog computing paradigm, in which edge devices are the parent of IoT devices. In this network, IoT devices can demonstrate a high degree of mobility, and they can establish trust relationships with various Edge parents.
Identity Privacy	In ASSURED, edge devices aggregate all IoT signatures of both successful and failed attestations so as to conceal the identify of the Prover IoT device. Only in the case of a failed attestation result, which depicts the possible compromise of a device, the authenticated root Verifier has the ability to link back the result to the initial proving device - without breaching the privacy of the remaining swarm nodes. This has been achieved through the integration of our Enhanced DAA protocol with introduced controlled-linkability and revocation features [44].
Attestation Evidence Privacy	In ASSURED, IoT Devices belonging to a swarm provide verifiable evidence to convince the parent edge device about the correctness of their operational state, while not revealing any configuration or control flow information.
Granularity of privacy	ASSURED produces swarm attestation evidence in different levels of granularity ranging from boolean answer of validating the entire network to exact identification of the compromised devices.

Table 2.1: Novel features of ASSURED Swarm Attestation.

2.1 High-level Overview of the Building Blocks and Workflow

In the ASSURED Swarm Attestation scheme, a **spanning tree structure** is adopted, where **parent Edge devices** can collect the attestation results of their **children IoT devices**, and forward the attestation reports based on these results to the Security Context Broker (SCB). While the first version of the scheme assumed a static network topology, as it will be elaborated throughout this deliverable, the second version was expanded to be able to support **dynamic network topologies** as well. The building blocks employed in the design and implementation of the ASSURED SA scheme are as follows:

- In order to achieve the **security, privacy and trust requirements** set forth by the ASSURED use case demonstrators, the scheme aims to preserve the identity and attestation privacy of the devices belonging to the swarm, while enabling **traceability and linkability only for compromised or malicious devices**. To this end, we employ ***group-based signature schemes***, enhanced with an ***enhanced Direct Anonymous Attestation (DAA) scheme***, which is capable of offering privacy-preserving traceability.

- In order to achieve the previously mentioned unforgeability and non-frameability properties, we employ ***short-term anonymous credentials (pseudonyms)*** that act as a non-repudiation means to guarantee that a malicious node is not able to forge an attestation result, and that the attestation result collected by a device cannot be denied.
- In order to guarantee an efficient and verifiable attestation launch, the initiation of the attestation is designed to include a ***nonce extracted from the ledger*** as part of the *createNonce* function of the attestation contract, and is signed using the ***DAA key binded to the underlying TPM-based Wallet***.

Leveraging the aforementioned building blocks, the ASSURED Swarm Attestation scheme has been designed and implemented as part of the ASSURED attestation toolkit. At a high-level, this scheme consists of five phases (more details on the underpinnings and crypto operations performed in each phase can be found in Section 4.3):

Setup: Initiated by the SCB (acting as the Verifier), it includes the establishment of network connectivity, topology, creation of the correct Attestation and DAA Keys (for Edge Devices) and short-term anonymous credentials (pseudonyms) for IoT devices. The former takes place during the Secure Enrollment phase [6], and involves the certification of the creation of the DAA Key with the Privacy CA, as well as the authentication with the Blockchain CA for getting the appropriate credentials to participate in the Blockchain infrastructure and to be able to download the appropriate attestation policies. The latter is performed by the IoT devices when the swarm is constructed under an Edge device, so the pseudonyms are created and registered under that Edge device. The output of this phase is the **creation and registration of all cryptographic primitives required to perform Swarm attestation**.

Request: Initiated by the SCB in order to request an authenticated measurement of the required attestation evidence (device configuration or software control flow) by the Runtime Tracer of each swarm device. In order to issue the challenge, the SCB first has to retrieve the nonce from the attestation policy on the ledger, sign the nonce, and send it to all swarm devices.

Attest: Executed individually by each swarm device in response to the attestation challenge. Each IoT device creates a message based on the traced attestation evidence, signs the message with its pseudonym, and sends it to its parent Edge device that certified the pseudonym. Upon the reception of these messages, each Edge device verifies the signatures. If this is successful, they verify the actual measurements as part of the overall attestation process. Depending on the attestation result per IoT device and the required privacy level, the Edge Device will create the appropriate aggregate result and sign it with its DAA key, and will also create its own attestation using the zero-knowledge proofs in the DAA scheme.

Report: Upon receiving the signatures from all its children IoT devices, each Edge Device will verify the signatures and aggregate only the valid ones. Therefore, if the check on an IoT signature is not successful, it will not be included in the aggregated signature. Then, each concatenates the group signatures created on the IoT Device pseudonyms together with the aggregated IoT signatures, and the final resulting swarm signature is sent to the SCB (acting as the Verifier) in order to verify if the swarm is in a trustworthy state.

Verify: Upon reception of the aggregate signature, the SCB checks whether all devices in the swarm are in a trustworthy state, and sends the attestation result to be recorded on the Blockchain ledger. Note that, when receiving the attestation result from each child IoT Device in the previous step, each Edge Device acts as a Verifier to perform the verification of the received measurements. Afterwards, based on the application requirements, the Edge device is able to follow one

of the following approaches: (i) achievement of complete privacy, (ii) anonymity based on the pseudonyms, and (iii) no privacy. Therefore, ASSURED enables each stakeholder to check the result on the Blockchain by leveraging the traceability features with the Group Tracer. Specifically, tracing of the IoT device can be achieved by the Edge Device that has knowledge of the long-term key corresponding to the certified short-term IoT device pseudonyms.

A detailed description of the first version of this scheme has been provided in D3.6 [22]. In Chapter 4, we provide details on the second version of this scheme, including the updates and new features outlined in Section 2.2.

2.2 Features and Updates on the ASSURED Swarm Attestation Scheme

The central motivation behind the design of the updates and new features implemented in the second version of the ASSURED Swarm Attestation protocol is the operation within the **Zero-trust paradigm**, (*"Never trust, always verify"*). Based on this principle, in the establishment of a communication channel between devices, no device is considered inherently trusted. In the context of the Swarm Attestation scheme, this means that **the devices participating in a Swarm should be able to verify the correctness of their configuration or operational state, without disclosing any personally identifiable information to the Verifier**, since based on the zero-knowledge principle, the Verifier cannot be considered trusted by default. As previously mentioned, this refers not only to **identity privacy** (i.e., the protection of the device identity), but also **attestation evidence privacy** (i.e., the protection of traced data used in the context of the attestation operation).

The notion of zero-trust is particularly important in the context of ASSURED, since the framework operates within organizations and SoS which are subject to strict security and privacy requirements. The concept of Zero-Knowledge Proofs was first introduced in [36], where a Prover makes a statement constituting a claim (e.g. about the correctness of its operational state) and Verifier issues a **challenge** to the Prover, who in turn has to respond with proof of the veracity of its statement. This proof has to satisfy the following conditions:

1. **Completeness.** If an honest Prover makes a statement in the form of a claim, the Verifier should be convinced of this fact by the Prover.
2. **Soundness.** If the Prover's statement is false, no malicious entity should be able to convince the Verifier that the statement is true, and vice versa.
3. **Zero-knowledge.** If the Prover's statement is true, the Verifier is not required to know anything else, except for the fact that the statement is true.

In the context of Swarm Attestation, in Chapter 5 we provide rigorous mathematical proofs on the achievement of these properties, in order to verify the theoretical correctness of the ASSURED final version of the produced Swarm Attestation scheme.

As it was previously mentioned, the achievement of attestation evidence privacy is important, since a malicious party who obtains configuration or control-flow information collected by the Runtime Tracer is able to infer information about various implementation aspects of the device, which may afterwards be used in order to compromise or hijack the device. These vulnerabilities

can have varying severity, and can be part of more complex attacks which take various sources of information into account. Some examples of attacks exploiting such vulnerabilities are as follows [1]:

1. Revealing the names of hidden directories, folder structure, and their contents, through a directory listing.
2. Obtaining access to source code files via temporary backups.
3. Obtaining database table or column names which are explicitly mentioned in error messages.
4. Direct access to highly sensitive information, such as credit card details.
5. Access to API keys, IP addresses, database credentials etc., which may be hard-coded into the source code.
6. Hinting at the existence or absence of resources, usernames, etc., via subtle differences in application behavior.

Another significant update to the ASSURED Swarm Attestation scheme presented in D3.6 [22] is the consideration of **dynamic network topologies**. In the first version, we considered a **static network topology** where the devices participating in the swarm and Verifier remain in static positions, and the network topology does not change at any time during the process. In a dynamic topology, we consider that it is possible for a device to change position, and each device has a **coverage range** within which communication with another device is possible. This introduces new issues that have to be addressed in the implementation of the attestation process. Specifically, if a healty device moves out of the coverage range of the Verifier (or becomes unresponsive for reasons such as turning to an idle state to reduce battery use), communication is not possible. However, this possibility may be exploited by a device that has been compromised by a malicious party, which may also deliberately stop transmitting in order to evade detection. Therefore, we cannot immediately assume that a device that moves out of range is compromised.

The consideration of dynamic topologies is particularly important in ASSURED in the context of the envisioned use cases, specifically with regards to the "*Smart Satellites*" use case, as detailed in Chapter 4. In Table 2.2, we summarize the features provided by the ASSURED Swarm Attestation enabler, including both the features that have been implemented and presented in D3.6 [22], as well as those implemented during the second reporting period, which are documented in this deliverable and have been integrated in to the ASSURED framework.

ID #	I want to <Action>,	so that <Reason>	**Implementation Status**
SA_1	As a system operator, I want to attest to the correctness of the configuration state or the execution flow of a software process on several devices simultaneously	I can ensure the correctness of all devices comprising the service graph chain throughout the operational lifecycle of the system, based on the (limited) computational capabilities of the edge devices.	Implemented in the first version of the SA protocol presented in D3.6 [22].

SA_2	As a system operator, when my device is part of a swarm, I want to attest to the correctness of the configuration state of my device, or the correctness of the execution flow of a software process running on my device, without disclosing any information about my identity	I can fulfill the identity privacy requirements of the system I belong to, and so that a malicious third party, or an "honest-but-curious" Verifier, cannot obtain information about my configuration state and leverage it in order to identify vulnerabilities and compromise my device.	Identity privacy has been implemented in the first version of the SA protocol presented in D3.6 [22].
SA_3	As a system operator, when my device is part of a swarm, I want to attest to the correctness of the configuration state/a control flow on my device, without sharing the collected system trace data used as attestation evidence with the Verifier	I can prevent any potentially malicious device from extracting any information from the traced data that can be used in order to obtain information that can be used to infer implementation data (i.e., memory flows) that can be used in implementation disclosure attacks.	Attestation evidence privacy is part of the second version of the SA protocol, presented in this deliverable. Implemented and integrated into the ASSURED framework.
SA_4	As a Verifier, in case of a failed Swarm Attestation process, I want to be able to trace the process to the device that caused the failed attestation	so that the source of any failure can identified, in order to enable the Attack Validation component to identify any new vulnerabilities or zero-day exploits so that the appropriate mitigation measures can be applied.	Traceability and linkability are presented in this deliverable, and have been finalized and integrated into the ASSURED framework.
SA_5	As a system operator, I want to have the capability to revoke a device that is part of the swarm, and which has been identified as the source of a failed attestation and constitutes a source of risk as a potentially compromised device (while also avoiding the revocation of a healthy device)	the integrity of the rest of the devices of the swarm, as well as the system as a whole, can be protected. Also, the rest of the (uncompromised) devices should continue their unimpeded operation.	The revocation feature is presented in this deliverable, and has been finalized and integrated into the ASSURED framework.
SA_6	As a system operator, I want to ensure that honest signatures can only be created by legitimate devices participating in a swarm	I can ensure that a malicious party cannot impersonate verified devices that have been legitimately and securely enrolled to the network.	Non-frameability and unforgeability have been taken into account in the first version of the SA scheme presented in D3.6 [22], and are also examined in this deliverable.

Table 2.2: Implementation status of ASSURED Swarm Attestation enabler.

Chapter 3

System/Threat Model & Listing of Security and Privacy Requirements

With regards to the system model employed in the ASSURED Swarm Attestation methodology, the first version of the protocol presented in D3.6 [22] considered a **static topology** of interconnected heterogeneous devices running mixed-criticality services, with various security and privacy requirements. In the final version of the protocol, we expand the system model to also support **dynamic (and hierarchical) network topologies**. As it was outlined in Section 2.2, supporting mobility of the Prover devices participating in a swarm is crucial, since it is required for the **efficient attestation of a multitude of devices whose communication path might be constantly changing** which results in different Verifier devices needed to be able to authenticate and validate the result of the attestation process. On top of that, **configuration and behavioural guarantees will need to be provided in a timely manner** so as to impede with the operational profile of the target application (as is the case for the envisioned "*Smart Satellites*" scenario).

Specifically, we consider a dynamic interconnected network of v Edge devices and k IoT actuators, following a hierarchical fog computing structure, as depicted in Figure 3.1. This architecture, which serves to distribute computational resources from the cloud to the network edge and closer to the data producers [47], provides novel opportunities in enabling the vision towards Next-Generation Smart Connectivity 'SoSeS envisioning the **evolution of such safety-critical SoS from local, stand-alone systems into safe and secure solutions distributed over the continuum from cyber-physical end devices, to edge servers and cloud facilities**.

In this context, a prominent milestone has been the integration of **Multi-Access Edge Computing (MEC)** capabilities that, as defined by the ETSIC MEC WG [1], has the potential to bring **extended network and computational resources closer to the edge so as to meet the strict latency requirements** of such (5G-enabled) vertical industries that at the same time are characterized by strict requirements as it pertains to fast service deployment times, dynamicity and trustworthiness but also exhibit different levels of security, privacy, trust and operational assurance goals, requirements and priorities. **This generates a clear trend towards distributed network models implemented through the Mobile Edge Computing (MEC) concept** [2]. According to this concept, the execution resources (compute and storage) are positioned at close proximity to the end devices and data generation sources. Edge and fog computing nodes coexist in a 5G fronthaul-backhaul infrastructure and support the mixed-criticality services [39] running either in the back-end cloud infrastructure or closer to the edge. **This denotes the decomposition of the envisioned service graphs into a mesh of "cloud-native" and "edge-running" mi-**

[1] https://www.etsi.org/technologies/multi-access-edge-computing

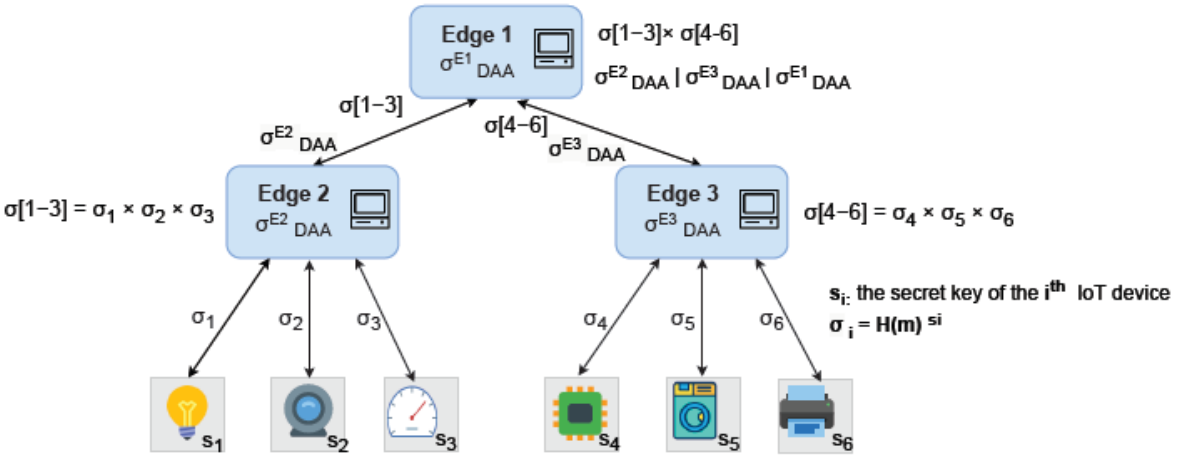

Figure 3.1: Overview of the Swarm Topology and Signature.

croservices each one with specific security, privacy, trust and safety objectives packaged on independent execution environments that need to be deployed over dynamically configurable, adaptive and higly available virtualized infrastructures capable of providing sufficient data connectivity and management of physical and virtualised network functions for a large number of distributed nodes. Essentially, the introduction of edge computing model alters the typical and simple-structured cloud-based connectivity model (access-core-cloud) to a mesh type model in which some functions must be executed at the edge part of the network and provide feedback to the attached end user devices, while portions of data can also be passed to the cloud. The complexity increases further by considering different types of edge nodes that may span from simple gateway servers to mini-data centers (DCs), thus having different connectivity requirements.

This type of emerging topologies necessitates the consideration of *hierarchical features* during a swarm attestation since multiple Verifiers (positioned at different layers) will need to validate the configuration software profile of their children nodes prior to relaying the information to the next node in the path. Considering also the **zero-trust** nature of the environment where such systems operate, **membership in a swarm might change over time (due to node mobility) and has to be established through the exchange of strong integrity guarantees on the trustworthiness of each comprised node**. As links to attesting devices are appraised as meeting a minim set of integrity requirements (through the employment of single-Prover attestation schemes as described in D3.3 [24]), these links are then included as **members of this trusted topology** and they have to be interpreted to statements on the integrity of the entire swarm topology in a privacy preserving manner. **This challenge has been identified by the IETF RATS as "*Trusted Path Routing Establishment*" [38] and is one of the gaps that the ASSURED Swarm Attestation aims to overcome**.

With regards to the mobility features of the devices, as depicted in Figure 3.2, we consider that both the IoT devices and/or the mobile Edge devices may move within or outside of the coverage range of a Verifier device. In this context, it is also possible for a Prover device to move to a different swarm, depending on its positioning. In the ASSURED Swarm Attestation scheme, we consider the following types of devices and components (as an extension of the system model already established in D3.6 [22]):

- **IoT Devices (D):** The devices belonging to the swarm, which consists of a group of heterogeneous sensors, actuators and Electronic Control Units (ECUs). Each IoT device is

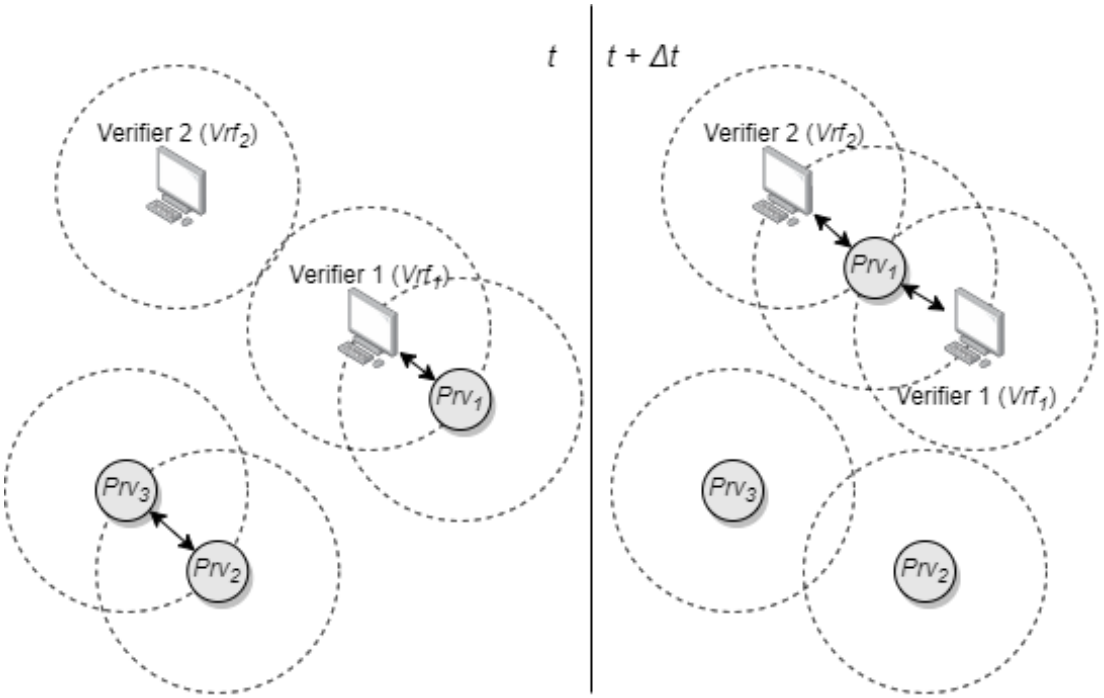

Figure 3.2: Dynamic network topology.

an untrusted resource-constrained device, such as the ones employed in the envisioned ASSURED use cases (e.g., the smoke sensors in the "*Smart Cities*" use case or the actual CubeSats as aforementioned in the context of "*Smart Satellites*"), each one of which is denoted by D_i for $i \in [1, k]$, where k is the total number of IoT devices in the swarm. Also, we assume that one IoT device is only connected to one edge device at a time (although the parent node can change due to enhanced mobility), and the IoT devices are not equipped with any additional (or specific) hardware for the provision of dedicated security controls.

- **Edge Devices (E):** Edge devices are untrusted powerful devices with large computational resources and storage capacity. These can be any kind of devices belonging to a network or a service graph chain, and are usually positioned over the MEC or a virtualized environment that can expose interfaces for communicating with IoT devices in a very efficient manner and over communication channels with minimum network delays. Each edge device is a combination of a host H_j and a Trusted Component (TC) (i.e., a TPM module) M_j, that is responsible for providing crypto the necessary trusted computing capabilities of executing in a trustworthy manner an attestation process; i.e., *provide Root-of-Trust capabilities for Storage, Reporting and Measurement*. Each Edge device is uniquely identified as E_j for $j \in [1, v]$, where v is the total number of Edge devices in the swarm. One E is a parent of a set of heterogeneous IoT devices and knows the legitimate state (i.e., "*golden hashes*" of the binaries) of its children. To this end, each Edge device authenticates its children and acts as their Verifier.

- **Verifier (V):** The Verifier is a device that **initiates the attestation process and aims to validate the trustworthiness of the entire swarm**. It can either be a device belonging to the network or an external third party. We consider that the Verifier knows the number of IoT devices in the swarm, and performs the verification in a legitimate manner, but might attempt to learn all possible information from the received attestation results (adopting an "*honest-but-curious*" mode of operation where it does not deviate from the intended execution of the

protocol but might be lured into leaking any sensitive information from the monitored Edge and IoT devices [35]). In other words, in order to fulfill the strict (aforementioned) identity and attestation evidence privacy requirements, no safety-critical or identifying information should be associated with any of the swarm devices unless there is a possible compromise detected through a failed attestation result.

- **Privacy Certification Authority (CA):** A trusted third party that acts both as an **Issuer** and a **Network Operator**. It is responsible for conducting the secure setup of the fog architecture, and is an integral part of the **secure enrollment process**, which was described in detail in D4.2 [6], and is responsible for verifying the creation of all relevant crypto primitives (such as the DAA key) and authorizing devices to join the network.
- **Runtime Tracer:** The **Runtime Tracer**, which is analyzed in detail in D3.4 [21] and D3.5 [25]), in the context of Swarm Attestation is part of the device's Trusted Computing Base (TCB) that enables the **secure introspection and monitoring of the host configuration and operational measurements that depict its runtime state to be attested**. The extracted traces are also ratified by the secure element, instantiated in the Edge device, and the verification result is signed by the Edge device's DAA Key. In case of traced "*quotes*" that does not match with the expected device state (as circulated by the Privacy CA to all respective Edge devices) then the root Verifier is notified as the only authorized entity to be able to link back this failed quote to the originating IoT device to initiate the appropriate raction strategy.

While the **hierarchical fog structure** considered in the first version of the ASSURED Swarm Attestation protocol, where Edge and IoT devices are considered part of a spanning tree, was originally designed to support static network topologies, it also captures the intricacies of **dynamic network topologies** considered in the second version. Specifically, it is important to consider the **mobility factor of the devices**, where a device may move away from one Edge device and move closer to another. In this case, towards fullfiling the need for **continuous authorization and authentication of the swarm devices**, the ASSURED SA scheme enables **connection of an IoT device with a different edge device, without requiring to recreate all the underlying cryptographic material**, in a seamless manner that does not disrupt the operation of the system.

As aforementioned, in the types of systems considered in ASSURED, **Edge devices typically have larger computational resources and storage capacity than IoT devices**. Therefore, they can act as **data distribution gateways** for data originating from children IoT devices, such as data collected from various sensors (for example, smoke detection sensors and surveillance cameras in the "*Smart Cities*" use case), across the data processing chain. The ASSURED Swarm Attestation scheme enables a Prover (Edge device) to convince a Verifier of the integrity of the aggregated attestation result of a swarm, based on true and signed attestation evidence shared by the enrolled IoT devices, **while safeguarding their identity and attestation evidence privacy**. In other words, the identity of an Edge and/or IoT device is not visible to the Verifier, unless the attested device is deemed compromised. Therefore, the ASSURED Swarm Attestation scheme should be able to return verified and anonymously signed attestation results, with **no assumption on the trustworthiness of the Edge/IoT devices or the Verifier**.

3.1 Adversarial Model

The security of swarm attestation depends on various factors, including the underlying cryptographic algorithms used and the protocols for exchanging messages and verifying the devices.

With regards to the considered **threat model**, which has already been defined in Section 2.2 of D2.1 [20], we consider the following types of adversaries on swarms:

- **Remote software-based adversaries** (Adv_{SW}) that aim to compromise IoT and Edge devices in the swarm, and disrupt the functionality the overall functionality by exploiting program vulnerabilities and injecting malicious code. Typically, the most prominent types of attacks that fall under this category are **code injection attacks** and **code reuse attacks**:
  - **Code Injection Attacks:** This type of attacks aims to exploit vulnerabilities such as out-of-bounds memory issues, and aims to inject malicious code while providing user input. This code will be stored and executed within the address space of the compromised application. In the context of Swarm Attestation, this can be addressed by applying CFA on the swarm devices.
  - **Code Reuse Attacks:** Here, the attacker does not inject new malicious code, but aims to alter the control-flow of the target device by interfering with the pointers corresponding to benign memory regions. Thus, by reusing existing code, the attacker creates *gadgets* (i.e., sets of low-level instructions) that simulate the attack path the attacker wishes to perform. Such attacks include *Return-Oriented Programming (ROP)*, *Jump-Oriented Programming (JOP)*, and *Counterfeit Object-Oriented Programming (COOP)*. Further information about these attacks is provided in D3.1 [17] and D2.1 [20].
  - **Resource Exhaustion:** An attacker exploits a resource exhaustion vulnerability in the software of a device in the swarm to disrupt its operations or to prevent it from performing attestation.
  - **Privilege Escalation:** An attacker exploits a privilege escalation vulnerability in the software of a device in the swarm to gain unauthorized access to the device or to sensitive information.
- **Network-based adversaries** (Adv_{NET}) that can forge, drop, delay, and eavesdrop the messages exchanged among two devices in the swarm. We consider a classical Dolev-Yao (DY) adversary [31] with full control over the communication channel between an edge device and an IoT device in the swarm, or between two edge devices, or between one edge device and the Verifier. However, *in line with other relevant schemes in the literature [10, 52], we assume the existence of a perfectly secure channel between the Host and the TC in an edge device that safeguards the correctness of the produced traces*. Therefore, the Adv_{NET} cannot intercept the interaction between the Host and the TC or use the TC as an oracle. Some examples of such types of attacks are as follows:
  - **Man-in-the-Middle (MITM):** If an attacker has control over the communication channel between two swarm devices, they can forge, drop, delay, or eavesdrop the messages exchanged between these devices.
  - **Replay Attacks:** An attacker captures messages exchanged between devices in the swarm and replays them at a later time to disrupt the swarm or to gain unauthorized access to the network.
  - **Sybil Attacks:** An attacker creates multiple fake devices in the swarm to manipulate the behavior of the swarm or to disrupt its operations.
  - **Eavesdropping Attacks:** An attacker passively listens to the communication between devices in the swarm to gain sensitive information, such as keys or authentication credentials.

  - **Spoofing attacks:** These aim to impersonate network traffic so that it appears to originate from a trusted and authorized source, and may include e-mail spoofing, IP address spoofing, DNS server spoofing, etc.
  - **Denial of Service (DoS)**: Performed by flooding the target device with extra unneeded requests to overload the system and reduce the availability of network or server services and resources.
  - **Malware**: Malicious software that can spread rapidly among connected devices, such as viruses, ransomware, and spyware.
  - **Compromised-key Attacks**: Refers to the exploitation of improperly implemented cryptographic protocols that lack forward and backward secrecy, where attackers can use compromised keys to obtain access to secure communication, or generate additional keys.
  - **Network Attacks on the TPM**: This is particularly notable in ASSURED, where TPM modules are embedded in the devices as Trusted Components (TCs). The host, namely the Trusted Software Stack (TSS) of the TPM can be hijacked in order to read, block, or modify the communication between the TPM and external Verifier. Thus, this can be seen as a subtype of MITM attacks.

- **Hardware adversaries** on swarm attestation include:
  - **Side-Channel Attacks:** An attacker exploits information leaked through side channels, such as power consumption, electromagnetic radiation, or timing, to extract secrets used for encryption and authentication or to bypass the attestation process.
  - **Physical Tampering:** An attacker physically modifies the hardware of a device in the swarm to bypass the attestation process or to compromise its security.
  - **Rogue Devices:** An attacker inserts a rogue device into the swarm to compromise its security or to manipulate the behavior of the swarm.
  - **Fault Injection:** An attacker injects faults into the hardware of a device in the swarm to disrupt its operations or to prevent it from performing attestation.

Assumptions: To defend against hardware-based attacks on swarm attestation, it is important to implement secure hardware design practices, such as secure boot, and in general prevent unauthorized modifications to the hardware. Additionally, it is important to use secure communication protocols and encryption to protect sensitive information from being extracted through side channels (an assumption been made for all ASSURED crypto primitives is been leakge resistant as this type of attack vectors falls outside the scope of the current research activities). It is also important to implement secure key storage mechanisms to prevent unauthorized access to secrets used for encryption and authentication. Finally, it is important to monitor the swarm for rogue devices and to detect and respond to physical tampering or fault injection attacks. However, note that **physical adversaries** (Adv_{PHY}) that are capable of extracting information from the TC are currently not considered in the ASSURED swarm attestation protocol.

In line with the state-of-the-art RA schemes, we assume software-only adversaries and we keep a Physical Adversary out of our current context. A Distributed Denial of Service (DDoS) attack is out of scope of this deliverable.

3.2 The Key Binding Problem

As has been extensively described in Deliverable D3.3 [24], one core innovation of the ASSURED attestation enablers is the offering of **local verification capabilities through the use of key restriction usage policies** which, in turn, enable the feature of zero knowledge. In this context, there is no need for sharing any attestation evidence from the Prover since the underlying Trusted Component (TC) is responsible for checking the correctness of the device prior to allowing it access to any securely stored crypto material (keys). Hence, the attestation result is encoded into a signature constructed with the use of an Attestation Key (AK) that is binded to a set of appropriate policies that govern the creation and usage of the key. More specifically, the AK is not released by the underlying TC to the host (for further processing as part of encryption and signing tasks) unless the policy is satisfied [44]. This is one of the core features of ASSURED that was introduced for **enabling efficient, scalable and privacy-preserving configuration integrity verification [28] based on the correct creation of the keys during the secure enrollment of the device into the overall network [6]**.

This means that the Root-of-Trust for the verification is enacted by the TC of the Prover which adds one extra layer of complexity during the (privacy-preserving) authentication and enrollment of the device into the system: Authenticating the correctness of a device prior to initiating the construction of the necessary keys (to support the entire lifecylce of the device binded to the appropriate key restriction usage policies) is not sufficient. This process needs to also verify that the device is equipped with a valid Trusted Component whose certificate has not been revoked for any reason. In what follows, we will expand on exactly this challenge - identified as "*Key Binding problem*" - considering all intricacies with the main one been not only on the verification that a (host) device is equipped with a valid TC but also on on the authentication between the TC and the host itself and elaborate on the steps taken in the ASSURED system model for resolving it.

We have to note that while this solution is applicable to any TC capable of managing such policies, for clarity we provide the details considering the use of a TPM as a trusted component based also on its instantiation for the experimentation of all envisioned ASSURED use cases.

A Trusted Platform Module (TPM)'s **Endorsement Key (EK)** is used in order to identify the TPM, while its **Attestation Key (AK)** is used in order to perform actions pertaining to attestation services. For instance, such operations may include **signing a set of Platform Configuration Registers (PCRs)**, **providing a timestamp**, or **certifying another TPM key**. The motivation behind using two separate keys is in order to protect user privacy, but it also raises the issue of **how to correctly bind these two keys together**. Key binding is indispensable towards building a revocation infrastructure. For example, consider a scenario where the Issuer is aware that a TPM's EK is compromised after it issues the credential to the TPM's AK. Key binding enables the Issuer to put the TPM's AK, correctly bound to its EK, into a revocation list. Therefore, key binding allows the correct execution of a revocation action. In this context, it should be ensured that **no honest user is revoked**, while **keys belonging to corrupt users should be revoked**. Thus, Key binding offers a "lightweight" key revocation solution.

Key binding is also required if **signatures need to be traceable**. Consider, for instance a rewarding service which allows users to earn incentives in an anonymous manner, by allowing a user to choose to link one or more transactions to their identity. If a user reaches a reward point, they want to link their signature to the original credential generation protocol. If the key binding property holds, the Issuer can confirm which TPM should obtain the reward. If the Issuer has a mismatching record binding an honest user's AK with a corrupted TPM's EK, as a result, the corrupted TPM will benefit from the honest TPM's contributions. As it is further elaborate in the

following Section 3.3, **traceability is a key requirement of the ASSURED Swarm Attestation scheme, and will be provided in the final design presented in this deliverable**.

The difficulty of solving the Key Binding problem stems not only from the "strong" privacy requirements of the TPM, but also from the strong trust assumptions that need to be made to capture the intricacies of the authentication between the TPM and the host. Specifically, a TPM is embedded in its host platform. Any communication between the TPM and the outside environment is performed via the host, and **the TPM can only respond to its host's commands**. Authentication between the TPM and the Issuer is also performed via the host. In the security threat model typically considered in trusted computing technologies (and is also adopted by ASSURED as outlined above), the host can potentially be an active attacker who can **read and modify messages sent from or to the TPM**, **block the communication between the TPM and external partner**, and can even **coordinate some rogue and undetected TPMs**. Because of the presence of the host with such physical proximity to the TPM and the lack of ability for the TPM to authenticate an external partner, it is very difficult for the remote partner to be convinced of a cryptographic binding between a TPM's AK and EK. Recently, Chen et al. [15] argued that no existing DAA scheme provides the Key-Binding property, and proposed a novel solution to this problem. The solution was also implemented in a real TPM module. We adopt this solution in our Enhanced DAA protocol in order to establish a secure authentication channel that binds the TPM's EK to its Attestation DAA key that leads to the correct revocation and linkability of swarm signatures [44].

3.3 Security & Privacy Requirements and Trust Assumptions

3.3.1 Security & Privacy Requirements

In Chapter 4 of D3.6 [22], we outlined the **security and privacy requirements** that need to be fulfilled by the ASSURED Swarm Attestation scheme. Here, we expand on these, and place them into the context of the newly implemented features. Also, in order to prove the correctness and soundness of the attestation protocol, we will perform **formal verification using formal logic and mathematically rigorous arguments**. This analysis, which will be presented in Chapter 5, will prove that our method achieves the below described requirements.

Note that, in order to conduct the mathematical proof, we employ the **Universal Composability (UC)** model [12], in which security follows the simulation-based paradigm, meaning that **a protocol is as secure as an ideal functionality, which performs the desired tasks in a way that is secure by design**. To this end, we aim to equate the **real-world network topology** in which the ASSURED Swarm Attestation protocol will be applied to an **ideal-world model**, which are both indistinguishable from each other from the perspective of a malicious third party. In this context, we aim to break down the attestation scheme into simpler building blocks with provable security, each one of which fulfils one or more of the envisioned requirements. The security definition in the UC model is given with respect to the high-level security properties given in Table 3.1, where we expand on the properties of the Swarm Attestation scheme outlined in D3.6 [22].

ID	Requirement	Description
SP1	Anonymity	Considering an external Verifier $\mathcal{V}$, it should be possible for honest Edge and IoT devices to conceal their identities from $\mathcal{V}$ by creating **anonymous signatures**. In other words, the Verifier should not be able to determine whether two such signatures originated from the same or different devices.

		In the case of IoT devices, anonymity is preserved through the use of **short-term anonymous key pairs (pseudonyms)** to create signatures, which should not be traceable to their source by the Verifier, except in the case of a failed attestation. In this case, **Linkability (SP7)** and **Traceability (SP2)** should be possible.
SP2	Traceability	A Verifier $\mathcal{V}$ who is part of the network should be able to trace aggregate signatures containing the overall attestation result back to the edge devices, while also allowing a parent edge device to link the signatures of its children back to the IoT devices.
SP3	Revocation	As an extension of **Traceability (SP2)**, it should be possible for the source of an attestation failure (e.g., an failed signature) to be traceable back to it source, in order to revoke a potentially compromised device. Conversely, the revocation method should be designed, so that the revocation of an honest device (i.e., a malicious party tricking the Verifier into believing an honest device is malicious) should not be possible.
SP4	Correctness	Signatures created in an honest and correct manner should also be considered valid. In addition, honest users should not share the same signing key.
SP5	Non-Frameability	It should not be possible for a malicious adversary to create signatures that may successfully impersonate an honest device, or link to honest signatures. In other words, it should not be possible for a malicious device to trick the Verifier into believing it is an honest device, or a device legitimately registered into the network.
SP6	Unforgeability	As an extension of **Non-Frameability (SP5)**, it should be infeasible for a malicious party to forge signatures, i.e., to produce attestation message signature pairs (μ, σ) that are accepted by the verification algorithm, without knowledge of the secret key of the users.
SP7	Linkability	Suppose that the Verifier is an **external device that is not an authenticated member of a network**. In this case, this Verifier should be able to audit the correctness of the swarm attestation process, and verify that the attestation process was executed by a correct device. Therefore, it should be possible to link aggregate signatures back to the edge devices, **without breaching their identity privacy**.
SP8	Provenance	A Verifier (Vrf) of a swarm is able to verify that the aggregate signature is based on the target devices participating in the swarm, by either certifying the identity of each device (without privacy considerations) or based on the correct number of group members (by checking the number of signatures included in the aggregate result) without being able to link these signatures to the source devices. Linkability (**SP7**) should only be enabled in case of a failed attestation.
SP9	Coalition Resistance	According to the Linkability (**SP7**) property, it should be possible to link a failed attestation to the device that caused it. However, a subset of the swarm devices may aim to circumvent this capability by colluding together to create signatures that cannot be traced to any of them. The designed scheme should not allow this to occur.
SP10	Exculpability	No member of the swarm (neither an Edge Device nor an IoT Device) should be able to produce signatures on behalf of the other swarm members.
SP11	Integrity of Swarm Attestation as a Whole	The aggregated attestation result should reflect the claimed integrity measurement during the attestation.
SP12	Integrity of exchanged communication data between devices	It should be ensured that the data exchanged between devices (such as attestation evidence sent from the IoT Devices to the parent Edge Device) is correct and legitimate, and threats such as the aforementioned MITM attacks should not be possible.
SP13	Compatibility with Static and Dynamic Networks	The Swarm Attestation scheme should be applicable to both static network topologies (where the Edge and IoT Devices remain in static positions), and dynamic topologies (where the Edge and IoT Devices may move in and out of the coverage range of the Verifier).

SP14	Evidence Authentication	We formalize the intuitive security requirement that an adversary $\mathcal{A}$ should not be able to pretend that an edge device E_j has a configuration cs_E satisfying the property that has to be attested (i.e., $cs_E \in CS$), when in fact $cs_E \notin CS$.
SP15	Configuration Privacy	The security requirement that the configuration cs_E of an edge device E_j should be kept private. For this requirement, both E_j and its embedded TPM have to be honest because E_j could always send cs_E to an adversary $\mathcal{A}$.

Table 3.1: Security Requirements of the ASSURED Swarm Attestation Scheme.

The privacy requirements of the ASSURED Swarm Attestation scheme, which are fulfilled by leveraging the newly implemented features that will be described in detail throughout the remainder of this section, are outlined in Table 3.2.

ID	Requirement	Description
PP1	Identity Privacy	In order to protect devices from malicious Verifying entities, no successful attestation result on a swarm of devices should be linked back to its device origin. Conversely, in order to protect the system from malicious devices, it should be possible to reveal the identity of a device that caused a failed attestation result. In this context, the ASSURED Swarm Attestation scheme should support three operational modes, namely Full Anonymity (**PP2**), Partial Anonymity (**PP3**), and No Anonymity (**PP4**).
PP2	Full Anonymity	During a Swarm Attestation process, it should be possible for the parent Edge Device to not merge the signed failed attestation result of an IoT Device in the IoT signature, so that the central Verifier should be able to understand that one (or more) IoT Devices failed with the execution of their attestation from the size of the aggregate signature (i.e., if the number of included signatures is less than the original size of the swarm). Thus, **only Verifiers with the appropriate authentication privileges should be able to link back a failed attestation result to the identity of the origin IoT Device**.
PP3	Partial Anonymity	In this operational mode, it should be possible for the parent Edge Device to merge all signatures into the aggregate signature (corresponding to both successful and failed attestations) through the use of short-term anonymous credentials (pseudonyms) used by the IoT devices to sign their individual attestation results. Thus, the identity of the IoT Devices will not be recorded in the attestation result on the ledger, but the linked pseudonym can only be linked by the parent Edge Device. **An authenticated Verifier that wishes to link back to the device identity needs to interact with the SCB to de-anonymize the Edge Device that performed the aggregation.**
PP4	No Anonymity	It should be possible to perform Swarm Attestation processes with no privacy considerations, where the parent Edge Device includes all (successful and failed) attestation results from the IoT Devices in the aggregate result, which is afterwards recorded to the ledger.
PP5	Attestation Evidence Privacy	It should be possible for the IoT Devices belonging to a swarm to be able to provide verifiable evidence that attests to the correctness of their operational state, while not revealing any configuration or control flow information to the parent Edge Device. Thus, the IoT Device should be able to utilize arithmetic circuits (zkSNARK) that require the attestation evidence as input, and only provide the attestation result as output, in order to be forwarded to the parent Edge Device.

Table 3.2: Privacy Requirements of the ASSURED Swarm Attestation Scheme.

3.3.2 Trust Modeling

To attest the trustworthiness of large-scale swarms in ASSURED, in the following, we outline the hardware components that need to be present in the **IoT Devices** and **Edge Devices**:

IoT Devices: To gurantee security of the attestation protocol, IoT devices involve software/hardware co-design, relying on **minimal hardware support** that consists of a **Read-Only Memory (ROM)** and a simple **Memory Protection Unit (MPU)** to guarantee code and memory isolation. Examples of research platforms with such capabilities include SMART [33], TrustLite [40], TyTan [9]. A commercial example includes ARM TrustZone technology [48] which provides security with hardware-enforced isolation built into the CPU, and it is integrated into today's Arm application processors and in the new generation Arm microcontrollers. Due to the attestation privacy requirements in ASSURED, we assume that IoT devices are equipped with a minimal trust anchor that enables control-flow tracing of the IoT execution at runtime. The **Runtime Tracer** has been analyzed in detail in D3.4 [21] and D3.5 [25].

Edge Devices: Edge devices in ASSURED are untrusted powerful device with large computational power and storage capacity. An Edge device is a combination of a Host and a **Trusted Component (TC)** that provides anonymity guarantees. In practice, ASSURED that each Edge device are equipped with a standardized **TPM** and utilizes **DAA protocol** to provide anonymous authentication of the Host without revealing additional information.

Chapter 4

Scalable Swarm Attestation Protocol with Enhanced Privacy Capabilities

As aforementioned, swarm attestation schemes aim to provide **scalable attestation solutions** that **verify the trustworthiness of large-scale networks in a more efficient way than attesting devices individually**. This is an important property for complex systems, as the ones envisioned in ASSURED, where the focus is to create **trust-aware service graph chains**, comprising heterogeneous devices (with different hardware capabilities), loaded with (possibly) **disparate software configuration profiles**. However, the endmost goal is not to simply reduce the communication overhead compared to the naive approach of applying many time single-Prover attestation tasks, but also enable an additional layer of **attestation intelligence and governance** that can allow Verifier devices to cope with the intrinsic characteristics of the service graph topologies that usually pervade these types of "Systems-of-Systems": (i) Such systems usually demonstrate a **high level of mobility** resulting in continuous changes occuring to the construction of the routing tree since they will need to establish new "*parent relationships*" with other nodes for relaying its communicated data, and (ii) The need to establish **federated trust between the devices in the swarm** so as to manage the *hierarchical topology* that usually characterize such systems where the device at each layer can act as the Verifier for its children or a *tree-based* structure where only the root node can act the root Verifier.

This type of features is one of the motivating factors for the enhanced version of the ASSURED Swarm Attestation mechanism that can support such **dynamic network topologies**, requiring **efficient and continuous authentication capabilities** (between the IoT device and the Edge device acting as the parent node in this particular time frame), and **flexible key management for enabling the verification of (signed) attestation reports in a hierarchical topology**: From intermediate nodes (acting as Verifiers) that might not have established a trust relationship with the Prover and need to query another trusted entity for the correctness of the presented crypto material or offload the verification process (thus, enabling a *verification-as-a-service* model).

Consider, for instance, the case of the "*Smart Satellites*" scenario, where multiple deployed satellites (operating in Low Earth Orbit (LEO)) need to exchange data with a Ground Station (GS) as part of the loaded (safety-critical or monitoring) application. This means that the Ground Station has only a limited time window for securely communicating with the satellite. Depending on the orbit altitude, the frequency and the capabilities of the Ground Station this time window can be in the order of few minutes. CubeSats normally are equipped with RF interfaces of limited bandwidth, resulting to a reduced data rate between the GS and the satellite. Therefore, in most cases, the **complete set of mission-related data cannot be transferred on a single pass over a GS**.

Compounding this issue, multiple Service Providers utilize the SatNOGS (see adjacent figure)

which is a network of multiple Ground Stations in different places across the globe, all connected through well protected cloud infrastructures. Having multiple Ground Stations in different places leads to an increased number of communication windows, thus, increasing the availability of a satellite. During a data transmission session, a Ground Station at the end of the communication window can inform the backhaul infrastructure regarding the amount of data successfully received from the satellite and which data still remain for receiving the complete amount of information defined as part of the specific mission. *The service that orchestrates the communication (running at the Mission Control Center (MCC) which acts as the root Verifier) can locate the next GS with an available (future) communication window and inform it to get ready for data reception*.

This translates to constantly changing the routing tree from the satellite to the MCC through different Ground Stations (or a set of multiple GS that form a path to the MCC), thus, aggravating the requirement for efficiently attesting the set of Ground Stations and the satellite prior to the establishment of a secure communication window. This requires having adequate **integrity guarantees on the correctness of each GS**, participating in the constructed data path in each session, and needs to be executed in a **timely manner** so that it does not strip any of the time available (during the communication window) for the transmission of the mission-related data.

On the other hand, consider the example in the "*Smart Aerospace*" use case where all on-board Electronic Control Units (ECUs) need to be able to provide operational information (collected during the duration of a flight) to the Ground Station (through the Secure Server Router (SSR) that can be acting as the Prover) accompanied with guarantees on the correctness of this data; *no ECU was compromised that may have altered the output of such operational data*. Thus, in such a flow, it is more **important to have strong assurance claims on the correct configuration and execution of all ECUs since there are no immediate time constraints** on the performance and collection of the required information when the airplane is on the ground. Thus, in this scenario, different ECUs will need to provide evidence based on the execution of either Configuration Integrity Verification (CIV) or Control-Flow Attestation (CFA) tasks but in a privacy-preserving manner so that no unauthorized entity or stakeholder, acting as Verifier, can fingerprint the type of aircraft based on the attested software configuration profile. This necessitates for the second feature implemented towards **attestation evidence privacy**.

Overall, the main motivation is to be able to have more efficient and scalable aggregate network attestation schemes, that while been able to integrate the attestation enablers already defined in D3.2 [18], will also enable ASSURED to make *intelligence decisions on how to adjust the execution of such security enablers in dynamic network topologies so as to identify the golden balance between converging security and safety without impeding the not the performance nor the privacy of the deployed devices*. The main crypto primitives employed towards achieving these two core functionalities are listed in the table (Table 4.1) below:

Crypto primitive	Description
Bilinear Signature Aggregation	In order for ASSURED to be able to accommodate **hierarchical system topologies** where parent Edge devices receive signed attestation results from their (children) IoT devices which can verify and forward them in an efficient and privacy-preserving manner to the upper layer node, it leverages aggregation capabilities.

Bilinear Signature Aggregation	More specifically, **Bilinear Aggregation Signatures** (Section 4.1.1) are employed whose ultimate purpose is to reduce the length of aggregate signatures in applications that are based in the construction and exchange of multiple signatures for safeguarding data (and attestation) integrity. Think, for instance, the aforementioned scenario in the context of "*Smart Satellites*": The satellite will transmit its data, alongside with a signature containing the attestation result on its correct configuration, to the Ground Station with which it can establish a communication window. The Ground Station, upon reception of this attestation attribute, it can verify the correctness of the software profile executed on the originator satellite and then construct an aggregate signature fusing also the result of its local attestation process (as executed by the underlying TCB) prior to forwarding it to the next Ground Station in the routing parth towards the MCC. Note that this scheme can be performed by any entity in the service graph chain, without the cooperation of the signers, including any untrusted party (not having established a trust relationship with any of the prevous signers, thus, enacting upon the zero-trust concept). As will also be described in the architectural details of the new SA scheme (Section 4.3), the use of bilinear signatures enables a rather efficient verification process.
DAA Signature	When an Edge device wishes to create a signature in a manner that enables controlled-linkability for one of its children IoT devices (*allowing the linkability of a provided signature back to the original signed only by authorized entities (root Verifier) and in the case of a failed attestation result*), the ASSURED Enhanced DAA scheme is employed [26] (Section 4.1.2). This enables the feature of *identity privacy* of the IoT devices, with linkability and traceability functionalities only in the case of a failed attestation and by the entity that has been authorized to manage the link token constructed during the Secure Enrollment phase.
Ring Signatures	In order to achieve *evidence authentication and evidence privacy*, we leverage the notion of ring signatures, which achieve the properties of *unforgeability* (i.e., it is not possible to create a signature without knowing one of the corresponding secret keys) and *anonymity* (i.e., it is not possible to deduce the identity of the device that created the signature). Therefore, we use DAA signatures to hide the identity of the Edge devices, and the anonymity property of ring signatures to achieve evidence privacy for the Edge Devices.

In what follows, we provide details on the aforementioned crypto enablers and their use in the context of the newly devised protocol.

4.1 Preliminaries - Dynamic SA Scheme Building Blocks

To enable a privacy-preserving and accountable swarm attestation approach, ASSURED Swarm Attestation adopts Direct Anonymous Attestation (DAA) to compute the device attestation in zero knowledge and relies on Bilinear Aggregation Signatures to efficiently aggregate the devices' signatures across the swarm.

4.1.1 Aggregated Signatures

As it was previously mentioned, the ASSURED Swarm Attestation scheme utilizes a hierarchical system model, where parent Edge Devices receive signed attestation results from their children IoT Devices, and afterwards aggregate their received signatures in order to forward them to the Verifier. In order to perform this aggregation, we employ the concept of **Bilinear Aggregation**

Symbol	Description
E_j	The j^{th} edge device
$\mathcal{M}_j$	TC that corresponds to the j^{th} edge device
$\mathsf{H_j}$	The host the corresponds to j^{th} edge device
$\mathcal{V}$	External Verifier
$\mathcal{I}$	Issuer
v	Number of edge devices in the swarm
tsk	TC's private key
r	Secret key used to generate ring signatures
C	The commitment of the device configuration
$CS = \{cs_1, \ldots, cs_n\}$	Set of acceptable configuration specifications
k_{ij}	The established Diffie- Hellman secret between E_j and D_j
s_i	The desired device D_i correct state
cs_E	The Edge device configuration
Q_j	E_j's public key known by the Privacy CA
(a, b, c, d)	E_j's credential created by the Privacy CA
T_j	E_j's public tracing key known by the Opener
T	The Opener's Tracing Key
q	A prime number that defines the order of cyclic groups, G_1, G_2
$g, g_1,\ g_2$ and g_3	Three generators of the groups G, G_1, G_2 and G_3 respectively
(x, y)	The Privacy CA private key
(X, Y)	The Privacy CA public key
(x_L, y_L)	The IoT device long-term key-pair
(x_p, y_p)	The IoT device short-term key-pair (i.e., pseudonym)
σ^L_{DAA}	The DAA signature of an edge device on the IoT device long-term public key y_L
σ^p_{DAA}	The DAA signature of an edge device on the IoT device short-term public key y_p
σ_i	The i^{th} IoT device signature on a message m_i using the short term secret key x_p
n_v	Attestation challenge
q	Prime number
ρ, r, s	Random numbers in $\mathbb{Z}_q$
k	Number of IoT devices in the swarm
P	Number of certified pseudonyms by edge device for each child IoT device
$\sigma_{[1-n]}$	The aggregated signature of n IoT devices
bsn	Random input in $\{0,1\}^*$ used for tracing and linking two DAA signatures
SPK	Signature based Proof of Knowledge
Function	**Description**
H	Hash function defined as $H : \{0,1\}^* \rightarrow G_1$
e	A pairing function defined as $e : G_1 \times G_2 \rightarrow G_3$ and $e(g_1, g_2) \rightarrow g_3$
Proofs	**Description**
π_{ipk}	Proof of the CA public key construction
π^1, π^2	Proofs of construction of Q_j and T_j respectively
π^{CA}_j	Proof of the E_j's credential construction
π_L, π_p	Proof of construction of y_L and y_p respectively

Table 4.2: Notation Summary

Signatures, whose ultimate purpose is to reduce the length of aggregate signatures in applications that use multiple signatures. Note that this scheme can be performed by anyone, without the cooperation of the signers, including any party untrused by the signers. It also does not impose an order on the signers, meaning that the order with which the IoT Devices send their attestation results to the parent Edge Device is irrelevant.

In the Bilinear Aggregation Signature scheme, consider that each IoT Device i creates a signature σ_i on a message m_i containing its attestation result, and afterwards sends the signed message to its parent Edge Device. Therefore, the parent Edge Device will eventually possess a set of n signatures $\sigma_1, ..., \sigma_n$ on messages $m_1, ..., m_n$ under public keys $PK_1, ..., PK_n$, respectively. In this case, the parent Edge Device will need to use a public aggregation algorithm in order to compress all n signatures into a single signature σ, whose length is the same as that of a signature on a single message. In order to verify the aggregated signature, a **verification algorithm** is employed, which receives all the original messages and public keys as input and

verifies whether the aggregate signature σ is valid.

The Bilinear Aggregation Signature consists of the following steps:

1. **Key Generation:** For a particular IoT Device, a random $x \leftarrow \mathbb{Z}_q$ is selected, and $w \leftarrow g^x$ is calculated. $x \in \mathbb{Z}_q$ and $w \in G$ are the private key and public key of the IoT Device, respectively.

2. **Signing:** For a particular IoT Device, given its public key w, its private key x, and the message $m \in \{0,1\}^*$ containing the attestation result, $h \leftarrow H(w,m)$ is computed, where $h \in G$, and $\sigma \leftarrow h^x$. The signature is $\sigma \in G$.

3. **Verification:** Given a IoT Device's public key w, a message m, and a signature σ, $h \leftarrow H(w,m)$ is computed. The signature is accepted as valid if $e(\sigma, g) = e(h, w)$ holds.

4. **Aggregation:** An index i is arbitrarily assigned to each IoT Device whose signature will be aggregated, ranging from 1 to n. Each IoT Device i provides a signature $\sigma_i \in G$ on a message $m_i \in \{0,1\}^*$. Then, $\sigma \leftarrow \prod_{i=1}^{n} \sigma_i$ is computed, and the aggregate signature is $\sigma \in G$.

5. **Aggregate Verification:** Given the aggregate signature $\sigma \in G$ for a set of IoT Devices (indexed as before), the original messages $m_i \in \{0,1\}^*$, and public keys $w_i \in G$, it is possible to verify the aggregate signature σ by computing $h_i \leftarrow H(w_i, m_i)$ for $1 \leq i \leq n$. The verification is successful if $e(\sigma, g) = \prod_{i=1}^{n} e(h_i, w_i)$ holds.

4.1.2 Direct Anonymous Attestation

In addition, in order to provide the required user-controlled linkability features that need to be present in the ASSURED Swarm Attestation scheme, we utilize an **Enhanced Direct Anonymous Attestation (DAA)** scheme, which has been designed in the context of ASSURED and has been described in detail in D3.2 [18]. In general, DAA is a platform authentication mechanism that allows **privacy-preserving remote attestation** of a device associated with a **Trusted Component (TC)**, instantiated here with a Trusted Platform Module (TPM). However, the scheme designed in ASSURED is agnostic to the type of TC used, and can be applied to TCs other than TPMs.

In ASSURED, we have designed an enhanced DAA scheme, which enables the Verifier to trace a failed attestation back to the IoT device that caused the failure. The ASSURED DAA scheme consists of five algorithms:

1. **SETUP:** In this phase, the system parameters must be chosen, and the Privacy Certification Authority (Privacy CA) needs to generate its keys. The system parameters and the Issuer's public keys are then published and made available to any node that needs to verify the validity of a signature.

2. **JOIN:** The device joins the group and obtains an Attestation Key Credential (AKC) for an ECC-DAA key created by the TC. The key can then be used to anonymously sign a message, or attest to data from this TC.

3. **SIGN:** Using the ECC-DAA key, a range of signing operations can be performed.

4. **VERIFY:** The DAA signature can be verified, in order to check whether it is valid or invalid.

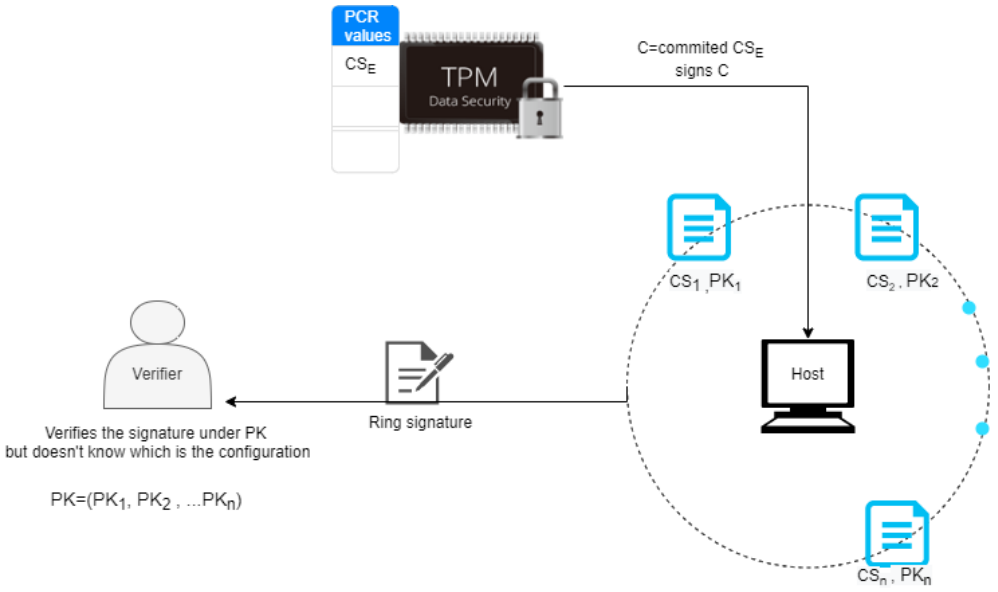

Figure 4.1: Property-based Attestation Framework enhanced with ring signatures

5. **LINK:** Checks two signatures to see of they are linked. It returns true or false, depending on whether they are linked or not.

Note that the DAA signature includes a **zero-knowledge proof** that is used in order to convince the Verifier that the signer possesses a valid membership credential, without divulging any information to the Verifier about the signer's identity. Thus, DAA is a protocol that provides the TC with the ability to sign its register values anonymously, while still convincing the Verifier that is possesses valid DAA credentials.

4.1.3 Ring Signatures

The concept of ring signatures was first introduced in 2001 by Rivest, Shamir and Tauman [51]. Ring signatures are a type of **anonymous digital signatures**, where the signer takes a number of public keys (referred to as the **ring**), as well as a secret key which corresponds to one of the public keys. When the signer outputs a signature, a Verifier can be convinced that a secret key corresponding to one of the public keys has been used for the signature, yet **the Verifier cannot tell which secret key is used**. The Verifier checks only the validity of the signature, but cannot know the identity of the actual signer. Ring signatures have many applications, such as e-voting and e-money.

In general, ring signatures have two main properties:

- **Unforgeability**: It is not possible to sign on behalf of a ring without knowing one of the associated secret keys.
- **Anonymity**: It is not possible to know the identity of the user that outputs a signature.

In the ASSURED Swarm Attestation scheme, we use the DAA signature to hide the identity of the Edge devices, and we need the ring signature anonymity property to achieve the evidence privacy of an Edge device instead of hiding its identity.

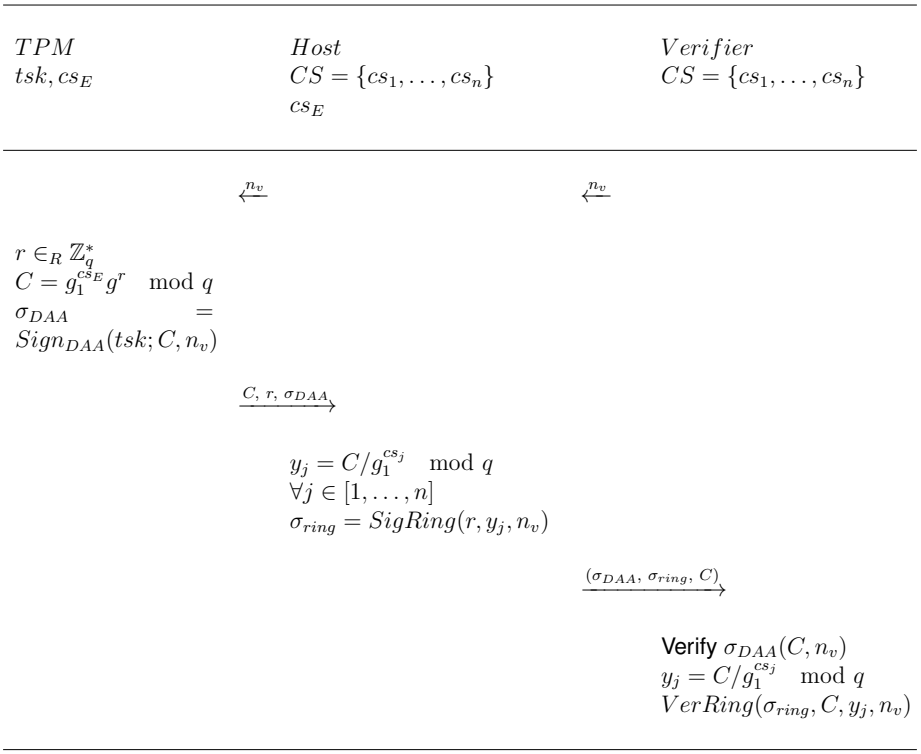

Figure 4.2: Property-Based Attestation (PBA) Sequence of Actions

To achieve evidence privacy and authenticity in the ASSURED Swarm Attestation protocol, we employ ring signatures where we define two main security requirements: **evidence authentication** and **evidence privacy**. While the former guarantees an **unforgeable binding between the platform and its evidence specification**, the latter provides the **non-disclosure of the evidence itself**. In the ASSURED Swarm Attestation protocol, these requirements are achieved through the use of a ring signature, i.e., evidence privacy results from the anonymity of the ring signature whereas evidence authentication is based on the unforgeability of the signature.

We denote the generation of a ring signature σ_{ring} on a message C with respect to the public key $y_{i\ 1\leq i\leq n}$ with private signing key r by $\sigma_{ring} := SigRing(r; y_i; C)$. Signature verification is denoted by $VerRing(y_i; \sigma_{ring}, C)$.

4.1.3.1 Ring Signature Scheme

- Key generation: Let γ be a security parameter. On input 1^γ, create g, q. A signer E_i for $(i = 1, \ldots, n)$ chooses $r_i \in_R \mathbb{Z}_q$, and computes $y_i = g^{r_i} \mod q$. It then outputs its public key y_i and the corresponding ring secret key r_i that will be used to create the ring signature.
- Signing algorithm $SigRing(r_j; y_i; C)$: A signer who owns the secret key r_j generates a ring signature on a message C with a public key list (g, p, y_i) for $(i = 1, \ldots, n)$, where $j \in \{1, ..., n\}$ by performing the following steps:
  1. Chooses $\alpha, c_i \in_R \mathbb{Z}_q$ for $i = \{1, \ldots, n\}$ and $i \neq j$.
  2. Computes $z = g^\alpha \prod_{i=1, i\neq j}^{n} y_i^{c_i} \mod q$.
  3. Computes $c = H(g|q|y_1| \ldots |y_n|C|z)$.
  4. Computes $c_j = c - (c_1 + \ldots + c_{j-1} + c_{j+1} + \ldots + c_n) \mod q$.

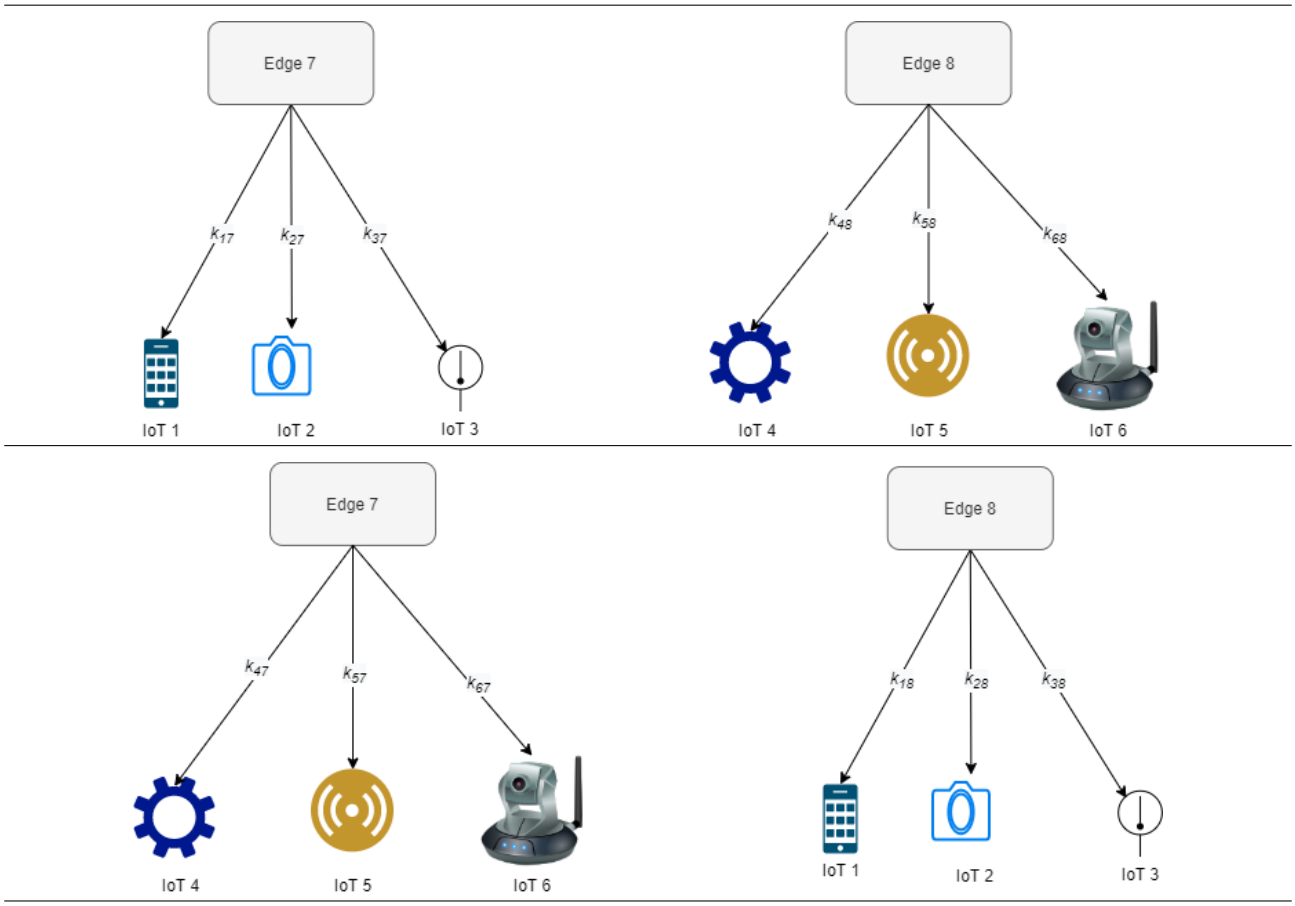

Figure 4.3: Dynamic Swarm Attestation setup, where k_{ij} is a shared secret between the i^{th} IoT device and the j^{th} Edge

5. Computes $s = \alpha - c_j r_j \mod q$.
6. Outputs the signature $\sigma_{ring} = (s, c_1, \ldots, c_n)$.

- Verification algorithm $VerRing(y_i; \sigma_{ring}, C)$: To verify that the tuple $\sigma_{ring} = (s, c_1, \ldots, c_n)$ is a ring signature on message C, check that $\sum_{i=1}^{n} c_i = H(g|q|y_1| \ldots |y_n|C|g^s y_1^{c_1}.y_2^{c_2} \ldots y_n^{c_n} \mod q)$.

4.1.4 Property-Based Attestation (PBA)

The key motivation behind **Property-based Attestation (PBA)** is to provide the ability to a computing platform to attest that a device fulfills the desired (security) requirements, the so-called 'properties', **without revealing the respective software and/or hardware information about the attested device**. ASSURED adopts PBA in order to provide **attestation evidence privacy** in Swarm attestation. In PBA, the host (Edge Device) creates a ring signature σ_{ring}, which is based on a TPM's DAA signature σ_{DAA} (to hide its identity) on the message C, which is a commitment to the edge device configuration cs_E . The TPM has to create and sign C, which it then opens towards the Host that is in charge of the creation of the ring signature, as shown in Figure 4.1. From the ring signature, the Verifier is convinced that the platform has been configured with one of the set of acceptable configuration specifications $CS = \{cs_1, \ldots, cs_n\}$, where each $cs_i \in \mathbb{Z}_q$, without

knowing the exact Edge device configuration. CS represents the list of correct configurations that have been circulated by the SCB (during the Edge device enrollment phase).

Note that in our protocol, the TPM is trusted by all parties, but its resources are restricted, and it can execute only a very limited set of instructions. The host is not trusted by the Verifier, hence the protocol has to protect evidence authentication against a malicious host. In fact, a malicious Host cannot be prevented from disclosing its own identity and configuration cs_E. Therefore, in order to achieve privacy, we have to assume that the host is honest.

Figure 4.2 presents a high-level overview of the PBA scheme from [16] that will be adopted in our our Swarm attestation Protocol.

4.2 Towards Dynamic Topologies with Device Mobility

Another important feature that we are investigating is the application of Swarm Attestation in the context of **dynamic network topologies**, as in [46], where the **mobility factor of devices is crucial and the verification of the IoT device outputs can be supported by any Edge device in the swarm** (instead of having only one root Edge device for each IoT device); i.e. we assume a dynamic tree-based topology structure). In this regard, we need to re-design the $setup$ phase of our swarm attestation protocol (including both device and swarm initialization phases) to be able to consider **continuous authorization and authentication of the swarm devices**. More specifically, whenever an IoT device gets connected with another Edge Device, **it should not be required to re-create all of the cryptographic material, but it should be possible for the Edge device to authenticate the IoT device using a predefined shared secret** as shown in Figure 4.3 that displays a Swarm of two Edge devices and six IoT devices. Each IoT device D_i for $i \in [1, 6]$ in the Swarm should agree on a shared secret with each Edge device E_j for $j = [1, 2]$. Whenever an IoT device wants to communicate with an Edge device, they authenticate each other via this established shared secret k_{ij}. More details on the construction of k_{ij} will follow in Section 4.2.1. Figure 4.4 showcases the change of topology of the Swarm and the use of the predefined k_{ij} to authenticate the Edge and IoT devices.

4.2.1 High-level Dynamic Swarm Attestation with Evidence Privacy

In this section, we highlight the techniques that we considered in order to achieve **Dynamic Swarm Attestation with evidence privacy**, starting from the Static Swarm Attestation design presented in D3.6 [22], where only identity privacy was considered. The concrete and detailed description of the whole protocol is presented in Section 4.3.

In the latest version of the ASSURED Swarm attestation scheme, the static Swarm Attestation scheme presented in D3.6 was modified as follows:

- The dynamic structure of the Swarm that is achieved by establishing a Diffie-Hellman (DH) authentication channel between each IoT and Edge device in the Swarm via a shared secret k_{ij}. k_{ij} is created once during the execution of the Secure Enrollment phase [6] between the i^{th} IoT device and the j^{th} Edge device in the Swarm for all $j \in [1, v]$.
- Attestation evidence privacy is achieved by creating a ring signature that hides the actual device configuration in a List CS, which represents the list of correct configurations circulated by the SCB (during the Edge device enrollment phase). The Edge device with its embedded TPM creates a ring signature σ_{ring} to hide its configuration cs_E and a DAA

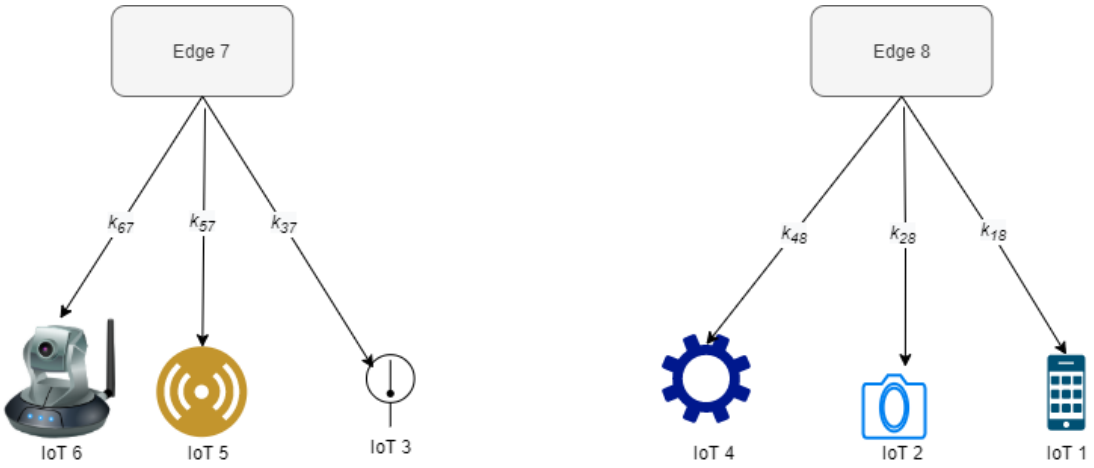

Figure 4.4: Dynamic Swarm topology

signature σ_{DAA} on the message C, which is a commitment to the edge device's current configuration cs_E. The TPM calculates and signs C, which then opens towards the Edge device (Host) that is in charge of the creation of the ring signature as shown in Figure 4.1.

- From the ring signature described in Section 4.1.3.1, the Verifier is convinced that the Edge device has been configured with one of the set of acceptable configuration specifications, $CS = \{cs_1, \ldots, cs_n\}$, where each $cs_i \in \mathbb{Z}_q$, without knowing the exact Edge device configuration.
- From the DAA signature described in Section 4.1.2, the Verifier is convinced that the commitment of the configuration is singed by a legitimate trusted platform TPM embedded in the Edge device without knowing the identity of the device. Therefore, the Verifier is able to verify the integrity of the device's configuration. However, the Verifier cannot learn neither the identity of the Edge device, nor its configuration.

Next, we provide a highlevel description of the Dynamic Swarm Attestation protocol interfaces (more details will follow in Section 4.3).

`Setup`: During the swarm initialization protocol, the SCB certifies the long term keys y_{L_i} for each IoT device D_i. D_i then creates a set of random short-term keys ($x_p \in \mathbb{Z}_q, y_p = g_2^{x_p}$), for each integer $p \in [1, \ldots P]$. These short-term keys will be later certified and kept in the records of the SCB. This certificate is a signature from the SCB that allows the IoT one-term keys (x_p, y_p) to be used by the IoT device D_i for creating signatures in the future. The desired device state s_i of each IoT device D_i is setup by the SCB and stored in the Blockchain so that it can be accessed by the edge devices.

`Edge-IoT Join`: During the Edge-IoT join inter-phase, each E_j agrees with every D_i in the swarm on a shared secret k_{ij} that is established through a secure DH channel as follows:

1. The edge device chooses a secret k_j.
2. For each IoT device D_i with public key y_{L_i}, the edge device E_j with a public key Q_j calculates $h_i = H(y_{L_i}, Q_j, s_i)$, where s_i is the expected correct state for the IoT device D_i.
3. The edge device sends $h_i^{k_j}$ to the IoT device D_i for some random secret k_j.

4. The IoT device chooses a random secret k_i, calculates $k_{ji} = (h_i^{k_j})^{k_i}$, and sends $h_i^{k_i}$ to the Edge device.

5. The Edge device calculates $k_{ij} = (h_i^{k_i})^{k_j}$.

6. The shared secret is $k_{ij} = k_{ji} = h_i^{k_j k_i} = h_i^{k_i k_i}$. Note that all the messages between E_j and D_i will be later exchanged via the established Diffie-Hellman channel with the agreed shared secret k_{ij}.

7. Finally, E_j and D_i keep $\{y_{L_i}, Q_j, k_{ij}\}$ in their records.

`Sign`: To create swarm signatures, the Edge Device E_j sends a freshly generated challenge n_v (extracted by the attestation policy from the Blockchain) to the IoT Device D_i, in order to initiate the Swarm Attestation process. In this context, IoT Devices follow the following steps:

1. The IoT Device loads its certified (by the SCB during the setup phase) short-term key (x_p, y_p) to create a signature.

2. The IoT Device signs a message s (current state) using it own short-term key certified key, the IoT-Device loads k_{ij} and computes the signature $\sigma = H(s, k_{ij}, n_v)^{x_p} \ mod\ q$.

3. The IoT Device sends its signature σ together with the signed message (s, k_{ij}, n_v) and the public short-term key y_p to E_j. E_j then checks that y_p has been previously certified by the SCB. If yes, it uses y_p it to verify the IoT signature σ, i.e. the Edge Device checks whether $e(\sigma, g_2) = e(H(s_i, k_{ij}, n_v), y_p)$ holds where s_i is the expected correct device state of D_i (i.e. $s = s_i$).

4. Each Edge Device collects all of its children (in the current topology) signatures using the above procedure, and aggregates its n children signatures $\sigma_1 \dots \sigma_n$, where n varies from one Edge to another. Each Edge Device computes the aggregate signature $\sigma_{[1-n]} = \sigma_1 \times \dots \times \sigma_n = H(m_1)^{x_1} \dots H(m_n)^{x_n}$ where $m_i = (s_i, k_{ij}, n_v)$.

5. Note that the verification of each σ_i is only performed by the Edge device E_j, which knows the components of the message m_i, whereas the aggregation verification will be re-checked by the root Verifier that doesn't have access to the exact message signed by the IoT Device (i.e. $m_i = (s_i, k_{ij}, n_v)$) since this reveals the actual s_i and k_{ij}. For the verification of the aggregation, the Verifier should only know $H(m_i)$ which perfectly hides s_i, since k_{ij} is unknown to the Verifier. In addition, the Verifier cannot link the state back to the actual signer (IoT device) when it signs different messages, since n_v adds freshness to the signed message every time.

6. The TPM commits the actual configuration of the device, then sends the commitment C to the host (corresponding Edge device) as shown in Figure 4.2.

7. The Edge device proceeds with creating a ring signature σ_{ring} to show that the committed configuration belongs to the set acceptable configuration specifications, CS.

8. The Edge Device together with its corresponding TPM create a DAA signature on its committed configuration C and the challenge n_v, then concatenates it together with the aggregated IoT signatures.

9. After aggregating the IoT Devices signatures of different Edge Devices, the final swarm signature $(\sigma_{ring}, \sigma_{DAA}, C, \sigma_{[1-k]}, H(m_i)_{i\in[1,k]}, \{y_p^i\}_{i\in[1,k]})$ is sent to root Verifier.

`Verify`: To verify this signature, the root Verifier checks whether the following equation holds:

$$e(\sigma_{[1-k]}, g_2) = e(H(m_1), y_1)e(H(m_2), y_2) \dots e(H(m_k), y_k)$$

, where k is the number of all the IoT Devices in the swarm. The Verifier verifies the DAA signatures of all the Edge Devices. The Verifier attaches the verification results to the challenge n_v used to create the swarm attestation in the Blockchain.

`Trace`: The tracing of Edge Devices is achieved by the Opener by using its tracing key to trace the DAA signatures. The tracing of IoT Devices is achieved by the parent Edge Device through the calculation of $e(\sigma, g_2) = e(H(s_i, k_{ij}, n_v), y_p)$ for each k_{ij}, knowing the challenge n_v used for creating the signatures. Only the parent Edge Device with the knowledge of k_{ij}, the expected state of the IoT device s_i and the session nonce n_v, can trace its child signature.

4.3 Architectural Details and Protocols of ASSURED Dynamic Swarm Attestation

Here, we present a high-level overview of the main phases of the ASSURED Swarm Attestation scheme, without focusing on the underlying cryptographic primitives. Details on all the phases of the scheme will be provided throughout the remainder of this section. Also, the notation used throughout the presentation of the protocol is presented in Table 4.2.

1. ***Setup Phase:*** This is a one-time offline procedure performed by the Privacy Certification Authority (CA) to guarantee the secure deployment of the devices in the swarm. We consider that each Edge Device E_j is equipped with a TC and pre-loaded with the expected legitimate state of its associated IoT devices, and since resource-constrained IoT devices do not support a TC, each IoT device is equipped with a long-term key whose public part represents the IoT device's identity. In addition, every time an IoT device joins another edge device, the setup phase has to be executed, and the edge device is expected to know its legitimate state. The setup phase consists of the following steps:
   (a) Once IoT devices are connected to a specific edge device, each IoT device D_i creates a set of pseudonyms (i.e., short-term keys) (x_p, y_p).
   (b) These pseudonyms will be certified by their parent E_j using the edge's DAA Key (i.e., an edge device creates a DAA signature on the public pseudonym y_p). The creation of an anonymous signature using pseudonyms enables the maintenance of a "hidden" identity of D_i when added to the aggregate signature so that the parent E_j can still identify that it belongs to its children IoT devices.
2. ***Attestation Phase:*** This phase entails the execution of all actions related to the generation of attestation evidence by the swarm devices, signature aggregation, and the subsequent creation of the attestation report. It consists of the following steps:
   (a) The attestation is initiated by the $\mathcal{V}$ that sends a challenge ch to one or more edge devices, which will then distribute it recursively to all the swarm devices.
   (b) Upon receiving this request, each D_i computes the integrity measurements producing an attestation result m_i and concatenates it with ch.
   (c) Once D_i signs the attestation result m_i using its pseudonym (certified by its parent edge device during the setup phase), it sends the signature to E_j.

(d) Upon receiving the signatures from all children IoT devices, each E_j verifies their attestation result and signatures and aggregates only the valid signatures signing them with its DAA Key.

(e) Afterwards, each edge device performs its attestation using the zero-knowledge proofs in the DAA scheme, creates its DAA signature, and concatenates it with the aggregated IoT signatures.

(f) After aggregating all the DAA and IoT devices' signatures from different Edge devices, the final swarm signature is sent to the Verifier.

3. ***Verification Phase:***

(a) After obtaining the Swarm Attestation report, the Verifier $\mathcal{V}$ verifies DAA signatures and checks the total number of IoT signatures to ensure that all IoT devices have been included in the report. Note that, in the ASSURED SA scheme, each E_j validates the attestation results and signatures received from IoT devices D_i.

(b) In the case of a non-valid DAA signature or a missing IoT signature, $\mathcal{V}$ claims that the swarm is not trusted and contacts the Opener to trace the compromised devices.

(c) To enable the Opener to trace the identity of the compromised edge devices (i.e., the DAA signatures), the traceability feature of the Enhanced DAA scheme enables the Opener to use a tracing key in order to identify the edge device with a failed DAA signature.

(d) In case an edge device is trusted but one of its children IoT devices is compromised, in order to identify the compromised IoT device, the Opener performs tracing through the parent edge device that knows the corresponding IoT long-term key linked to the certified IoT pseudonyms.

(e) When an edge device is compromised, the Opener cannot verify the trustworthiness of the attestation evidence of the IoT devices as only the parent edge devices have prior knowledge about the expected legitimate state of IoT devices. However, the verification of the IoT signatures is still possible, thanks to the construction of the pseudonyms, which does not allow an Edge device to sign on behalf of its children.

In the following, we elaborate on all the aforementioned phases of the ASSURED SA protocol.

4.3.1 Setup Phase

The setup phase encompasses the **formal key generation of the Privacy Certification Authority (Privacy CA)**, the **enrollment of edge devices** E_j when the DAA Attestation Key is generated (and certified by the Privacy CA), and the **enrollment of IoT devices** D_i when long-term and short-term key pairs are created by the IoT devices and certified by the SCB. Specifically, the setup phase starts with the Privacy CA generating her key pairs $x, y \leftarrow \mathbb{Z}_q$, sets $X = g_2^x$ and $Y = g_2^y$. The Privacy CA proves that this key is well formed and registers its key (X, Y, π_{ipk}).

4.3.1.1 Edge Device Enrollment

The Edge Device Enrollment process starts from an edge device E_j, where $j \in [1, v]$, that sends an enrollment request to the Privacy CA for joining the target service graph chain. The Privacy CA chooses a fresh nonce ρ and sends it to E_j, who forwards it to its TPM $\mathcal{M}_j$. Then, the TC

chooses a secret key $tsk \leftarrow \mathbb{Z}_q$, sets its public key $Q_j = g_1^{tsk}$, and computes π_j^1 as a proof of construction of Q_j. Once $\mathcal{M}_j$ sends (Q_j, π_j^1) to the Privacy CA, it verifies π_j^1 to check whether the Edge device is eligible to join, i.e., the Edge device DAA key has not been registered before. Upon a successful validation, the Privacy CA picks a random $r \leftarrow \mathbb{Z}_q$ and generates E_j's DAA credential (a, b, c, d) by setting $a = g_1^r, b = a^y, c = a^x Q_j^{rxy}$ and $d = Q_j^{ry}$. In addition, the Privacy CA generates a proof π_j^{CA} on the credential construction.

4.3.1.2 Generating the Tracing Keys by the Opener

Each TPM $\mathcal{M}_j$ sets $T_j = g_2^{tsk}$ and computes π_j^2 as a proof of construction of T_j. $\mathcal{M}_j$ sends $(T_j, Q_j, \pi_j^1, \pi_j^2)$ via E_j to the Opener. The Opener verifies π_j^1 and π_j^2 and makes sure that T_j and Q_j link to the same TC by checking if the following holds:

$$e(Q_j, g_2) = e(g_1, T_j) \tag{4.1}$$

To convince the Opener that the key T_j correct, $\mathcal{M}_j$ may also send to the Opener the DAA credential of Q_j issued by the Privacy CA. After receiving all Edge devices T_j, for all $j \in [1, v]$, and their corresponding public keys Q_j, the Opener sets its tracing key T as follows: $T = \{T_1, T_2, \ldots, T_v\}$ and keeps the (Q_j, T_j) pairs in its records for all $j \in [1, v]$.

4.3.1.3 IoT Device Enrollment

Each IoT device is equipped with a long-term key pair (x_L, y_L) whose public part $y_L = {g_1}^{x_L}$ represents the IoT device's identity and is certified by a trusted authority. Let π_L represent a proof of construction of $y_L = {g_1}^{x_L}$. The enrollment of IoT devices consists of the following steps:

1. The SCB certifies each of the IoT long-term key by creating a signature σ_{DAA}^L on y_L.
2. Once the long-term key is certified, each IoT device creates short-term random keys ($x_p \in \mathbb{Z}_q, y_p = g_2^{x_p}, \pi_p$), for each integer $p \in [1, \ldots P]$ where π_p is a proof of construction of y_p and P is the total number of pseudonyms that will be certified by the SCB for each IoT Device. The certification on the short-term keys will be sent back to the IoT Devices.

In ASSURED, the setup phase is an initial offline procedure that is performed only once before the attestation itself is performed. In particular, this phase aims to **securely deploy swarm devices** and is performed by the **Privacy Certification Authority (Privacy CA)**. During this phase, it is ensured that each edge device is equipped with a Trusted Component. The Edge devices E_j establish a secure DH channel and share a secret k_{ij} with every IoT device D_i in the swarm as follows:

1. The Edge device chooses a secret k_j.
2. For each IoT device D_i with public key y_{L_i}, the Edge device E_j with a public key Q_j calculates $h_i = H(y_{L_i}, Q_j, s_i)$ where s_i is the expected correct state for the IoT device D_i.
3. The Edge device sends $h_i^{k_j}$ to the IoT device D_i for some random k_j.
4. The IoT device chooses a random secret k_i, calculates $k_{ji} = (h_i^{k_j})^{k_i}$ and sends $h_i^{k_i}$ to the Edge device.
5. The Edge device calculates $k_{ij} = (h_i^{k_i})^{k_j}$.

6. The shared secret is $k_{ij} = k_{ji} = h_i^{k_j k_i} = h_i^{k_i k_i}$. Note that all the messages between E_j and D_i will be later exchanged via the established Diffie-Hellman channel with the agreed shared secret k_{ij}.

7. Finally, E_j and D_i keep $\{y_{L_i}, Q_j, k_{ij}\}$ in their records.

4.3.2 Attestation Phase

The Swarm Attestation process is initiated by a Verifier $\mathcal{V}$ who sends a challenge n_v to an Edge device, which will then distribute it recursively to all devices in the swarm. The first devices that perform attestation are the IoT devices, which sign their attestation results and report them to their parent edge devices.

4.3.2.1 IoT Device Signature

When an IoT device receives an attestation challenge n_v, it will perform the attestation process, concatenate the attestation result with the challenge n_v, and sign the attestation response (i.e., message $m_i = (s, k_{ij}, n_v)$) using its own pseudonym (i.e, short-term key previously certified by the SCB during the enrollment procedure in the setup phase). The the IoT device computes its signature as follows:

$$\sigma_p = H(m_i)^{x_p} \mod q \tag{4.2}$$

Next, the IoT device sends its signature σ together with y_p to its parent edge, which will check that y_p is certified by the SCB and verify the signature σ and determine whether the following holds:

$$e(\sigma, g_2) = e(H(m_i), y_p) \tag{4.3}$$

Assume that n IoT children devices successfully pass the signature verification. In this case, the edge device aggregates its n children's signatures $\sigma_1, \ldots \sigma_n$ in an aggregated signature $\sigma_{[1-n]} = \sigma_1 \times \ldots \times \sigma_n = H(m_1)^{x_1} \ldots H(m_n)^{x_n}$. When the attestation of an IoT device is unsuccessful or the attestation of a given IoT device is missing, then the edge device will not consider that IoT device in the aggregation.

4.3.2.2 Edge Device Signature & Traceability

The creation of a traceable DAA signature starts with an edge device E_j that decides a basename bsn as a random string $\{0,1\}^*$ to trace and control linkability between two DAA signatures. To sign a message μ with respect to the basename bsn, the edge device re-randomizes its credential by choosing a random $\tau \leftarrow \mathbb{Z}_q$, sets $(a', b', c', d') \leftarrow (a^\tau, b^\tau, c^\tau, d^\tau)$, and then sends $(\mu, \mathsf{bsn}, \tau)$ to the $\mathcal{M}_j$. Next, $\mathcal{M}_j$ checks that $b' = b^\tau$ and $d' = d^\tau$, sets $nym = (\mathsf{bsn})^{tsk}$, and calculates a Signature based Proof of Knowledge SPK on the TPM's secret key tsk and the device's credential on (μ, bsn) as

$$SPK\{tsk : nym = H(\mathsf{bsn})^{tsk}, d' = b'^{tsk}\}(\mathsf{bsn}, \mu) \tag{4.4}$$

The DAA signature is $(a', b', c', d', SPK, nym)$.
The TPM commits the actual configuration of the device, then send the commitment C to the host (corresponding Edge device) as in Figure 4.2. The Edge device proceeds with creating a ring signature σ_{ring} to show that the committed configuration belongs to the set of acceptable configuration specifications, CS. The Edge Device together with its corresponding TPM create a DAA signature on its committed configuration C and the challenge n_v, then concatenates it together with the aggregated IoT signatures.

Each Edge Device aggregates its the IoT Device signatures and creates its own DAA on the commitment of its configuration C and ring signature as previously discussed. The final swarm signature is $(\sigma_{ring}, \sigma_{DAA}, C, \sigma_{[1-k]}, H(m_i)_{i\in[1,k]}, \{y_p^i\}_{i\in[1,k]})$, where k is the total number of IoT devices in the swarm, is sent to the root Verifier.

4.3.3 Verification Phase

After receiving an attestation report, the Verifier checks the DAA signature of each Edge device $\sigma = (a', b', c', d', nym, SPK)$ on a message μ w.r.t. a basename bsn that is published with the signature. In particular, the Verifier verifies SPK w.r.t. (μ, bsn) and nym, and checks that $a' \neq 1$, $e(a', Y) = e(b', g_2)$, $e(c', g_2) = e(a'd', X)$. In order to check the aggregate result, the Verifier checks if the following holds:

$$e(\sigma_{[1-k]}, g_2) = e(H(m_1), y_1)e(H(m_2), y_2)\dots e(H(m_k), y_k) \tag{4.5}$$

where k is the total number of IoT devices in the swarm. If the aggregate result has less than k IoT aggregated signatures or includes any failed signature, then the Verifier claims that the swarm attestation has failed. Further, the Verifier can interact with the Opener to initiate the tracing of the device with a failed attestation.

4.3.3.1 Link

When a Verifier has access to different swarm attestation reports, it can check if the DAA signatures originate from the same edge devices or not. Specifically, the Verifier checks if both signatures are signed under the same basename known by the Verifier.

More formally, given $\sigma_1 = (a'_1, b'_1, c'_1, d'_1, nym_1, \pi_1)$ and $\sigma_2 = (a'_2, b'_2, c'_2, d'_2, nym_2, \pi_2)$ on a message μ w.r.t. a basename bsn. In this setting, the output is 1 when both signatures are valid and $nym_1 = nym_2$, otherwise the output is 0.

4.3.3.2 Tracing/Opening

In ASSURED, traceability consists of two levels: (i) **tracing an aggregate attestation signature to the edge device** and (ii) **tracing a failed attestation to the origin IoT device**. We add a traceability requirement to the existing DAA scheme [11] in order to obtain a novel **Traceable DAA** protocol that meets our Swarm Attestation security and privacy requirements. The first level of swarm attestation traceability relies on the traceability of the DAA signature that can be performed by the Opener using its tracing key.

Specifically, starting with a DAA signature σ_{DAA} with a corresponding link token $nym = H(\text{bsn})^{tsk}$, the Opener uses its tracing key $T = \{T_1, T_2, \dots, T_v\}$ to check the following equation:

$$e(nym, g_2) = e(H(\text{bsn}), T_j) \;\; \forall\, j \in [1 - v] \tag{4.6}$$

A successful verification allows the Opener to recover the TC public key Q_j corresponding to T_j in its records, as it was previously explained during the setup of the Tracing Keys by the Opener.

Once the identity of the edge device is recovered, it is easy for the Edge device to recover any of the long-term public keys of its children that are certified by this Edge device relying on its records. In particular, The tracing of IoT Devices is achieved by the parent Edge Device through the calculation of $e(\sigma, g_2) = e(H(s_i, k_{ij}, n_v), y_p)$. Only the parent Edge Device with the knowledge

of k_{ij}s for all the IoT devices, the expected state of the IoT devices s_i, and the session challenge n_v, can trace its child signature. Note that in dynamic network topologies, only the Edge device that acted as the parent of the IoT device when creating a signature can trace this signature back to the IoT device with the particular knowledge of k_{ij}.

4.3.3.3 Revocation

Any **Revocation Authority (RA)**, which may also be the Opener, can remove misbehaving edge devices without revealing the edge device's identity. The motivation behind the revocation scheme is to be able to deactivate any DAA credential of an edge device, created during its enrollment [44]. We assume that a Revocation Authority RA has access to a set of revoked DAA keys `KRL`. The Revocation Authority checks that the DAA key used to create the DAA signature is not in the revocation list. This check is performed by simply verifying that the following holds: $\forall\, tsk^* \in \texttt{KRL},\ nym \neq H(bsn)^{tsk^*}$.

The ASSURED dynamic protocol also supports revocation of the IoT devices by their parent edge device. We assume that the Edge device has access to a set of IoT revoked keys, referred to as the **IoT-Key Revocation List** `IoT-KRL`, based on which the Edge device checks that each of its children IoT signatures were not produced by any key in `IoT-KRL`. Relying on the trust of the edge devices and the accuracy of the updated `IoT-KRL`, Edge devices would also be able to correctly revoke the IoT devices whose keys are in `IoT-KRL`.

Chapter 5

Security Analysis of the ASSURED Swarm Attestation Protocol

5.1 Swarm Attestation Protocol Security Analysis

5.1.1 Methodology

We employ the Universal Composability (UC) model, based on which we aim to equate the real-world network topology to an ideal-world model, both indistinguishable to each other from the perspective of an adversary, in which it is possible to break down the proposed scheme into simpler building blocks with provable security. Specifically, the security definition of the ASSURED Swarm Attestation protocol is given with respect to an ideal functionality $\mathcal{F}$. In UC, an environment $\mathcal{E}$ should not be able to distinguish, with a non-negligible probability, between the two worlds:

1. The *real world*, where each party E_j or D_i (corresponding to the Edge and IoT devices, respectively) executes its assigned part of protocol Π. The network is controlled by an adversary $\mathcal{A}$ that communicates with the environment $\mathcal{E}$.

2. The *ideal world*, in which all parties forward their inputs to a trusted entity, called the ideal functionality $\mathcal{F}$, which internally performs all the required tasks and creates the parties' outputs in the presence of a simulator $\mathcal{S}$. This essentially translates to the Edge and IoT devices sending their outputs to $\mathcal{F}$, which is then able to perform all operations in a trustworthy manner, and provide the compiled output.

5.1.2 Ideal Functionality Algorithms for the Dynamic Swarm Attestation with Evidence Privacy

We define the algorithms that will be used inside the ideal functionality $\mathcal{F}$. These algorithms will be called by $\mathcal{F}$ in our security model to provide key generation, DAA signatures, and ring signatures with their verification (as in real world protocol) on behalf on honest parties, as well as additional algorithms that are used by $\mathcal{F}$ to identify the signer in our model for linking and revocation purposes. These algorithms are defined as follows:

Kgen(1^λ): A probabilistic algorithm that receives as input a security parameter λ and generates keys tsk and dsk for honest TCs and IoT devices respectively. This is part of the SETUP phase

for certifying the correct creation of the DAA key of the edge device and the pseudonyms of IoT devices.

Sign: A probabilistic algorithm used for honest TCs and IoT devices in order to create the type of signature required by the corresponding operation. It consists of the following sub-algorithms:

1. $\text{sig}_{\text{DAA}}(tsk, \mu, \text{bsn})$: On input of a secret key tsk, a message μ and a basename bsn, it outputs a DAA signature σ_{DAA}.
2. $\text{sig}_{\text{ring}}(C, r)$: On input of a random secret key r, a message C, it outputs a ring signature σ_r.
3. $\text{sig}_{\text{IoT}}(dsk, m)$: On input of a secret key dsk for an IoT device D_i and a message m, it outputs a signature σ_i.
4. aggregate($\sigma_1, \sigma_2, \ldots, \sigma_c$): On input $\sigma_1, \sigma_2, \ldots, \sigma_c$, it outputs $\sigma_{[1-c]}$.

Verify: A deterministic algorithm that is used in the VERIFY interface, and outputs a binary result on the correctness of a signature. It consists of two sub-algorithms:

1. $\text{ver}_{\text{DAA}}(\sigma_{DAA}, \mu, \text{bsn})$: On input of a signature σ_{DAA}, a message μ and a basename bsn, it outputs $f = 1$ if the signature is valid, $f = 0$ otherwise.
2. $\text{ver}_{\text{ring}}(C, \sigma_r)$: On input of a ring signature σ_r, and message C, it outputs $f = 1$ if the signature is valid, $f = 0$ otherwise.
3. $\text{ver}_{\text{IoT}}(\sigma_i, m)$: On input of a signature σ_i, a message m, it outputs $f = 1$ if the signature is valid, $f = 0$ otherwise.

Link: $(\sigma_1, \mu_1, \sigma_2, \mu_2)$: A deterministic algorithm that will be used in the LINK interface, which checks if two signatures originate from the same device. In case of DAA signature linkability checks, an extra parameter bsn is required as an input. It outputs 1 if both σ_1 and σ_2 were generated by the same device, and 0 otherwise.

Identify: A deterministic algorithm that will be used to ensure consistency with the ideal functionality $\mathcal{F}$'s internal records, by connecting a signature to the key used in order to generate it. It consists of two sub-algorithms:

1. $\text{identify}_{\text{DAA}}(tsk, \sigma_{DAA}, \mu, \text{bsn})$: It outputs 1 if a key tsk was used to produce a signature σ_{DAA}, and 0 otherwise.
2. $\text{identify}_{\text{IoT}}(dsk, \sigma_i, m)$: It outputs 1 if a key dsk was used to produce a signature σ_i, and 0 otherwise.

5.1.3 Universal Composability Security Model

We now explain the interfaces of the proposed ideal functionality $\mathcal{F}$ in the UC framework based on the functionality originally proposed in [11]. The UC framework allows us to focus the analysis on a single protocol instance with a globally unique session identifier sid. $\mathcal{F}$ uses session identifiers of the form $sid = (\mathcal{I}, sid')$ for some issuer $\mathcal{I}$ and a unique string sid'. In the real-world, these strings are mapped to the issuer public key, and all parties use the sid to link their stored key material to the particular issuer. We define the edge device JOIN, IoT device JOIN, and SIGN

sub-session identifiers $jsid, jsid'$ and $ssid$, respectively, to distinguish any join and sign sessions that might run in parallel.

We define two "macros" to determine if a secret key key is consistent with the internal functionality records or not. This is checked at several places in the ideal functionality interfaces and also depends on whether key belongs to an honest or corrupt party. The first macro CheckkeyHonest is used when the functionality stores a new key that belongs to an honest party, and checks that none of the existing valid signatures are identified as belonging to this party. The second macro CheckkeyCorrupt is used when storing a new key that belongs to a corrupt party, and checks that the new key does not break the identifiablity of signatures, i.e., it checks that there is no other known key^*, unequal to key, such that both keys are identified as the owner of a signature. Both functions output a bit b, where $b = 1$ indicates that the new key is consistent with the stored information, whereas $b = 0$ signals an invalid key. Finally, we assume that the host H_j represents the same entity as E_j. Thus, $\mathcal{M}_j$ and E_j correspond to the TC and the host respectively in an edge device j.

SETUP

1. On input (SETUP, sid) from issuer $\mathcal{I}$, output (SETUP, sid) to $\mathcal{S}$.
2. On input (ALG, sid, Sign, Verify, Link, Identify, Kgen) from $\mathcal{S}$,
   - Check that Verify, Link and Identify are deterministic.
   - Store the algorithms and output (SETUPDONE, sid) to $\mathcal{I}$.
   - Define a random static list `ConfigurationList` that hides the hash of correct configurations

Edge-JOIN

1. On input $(\text{JOIN}, sid, jsid, \mathcal{M}_j)$ from host E_j, create a join session record $\langle jsid, \mathcal{M}_j, E_j \rangle$ and output $(\text{JOINPROCEED}, sid, jsid, \mathcal{M}_j)$ to $\mathcal{I}$.
2. On input $(\text{JOINPROCEED}, sid, jsid)$ from $\mathcal{I}$, output $(\text{JOINCOMPLETE}, sid, jsid)$ to $\mathcal{S}$.
3. On input $(\text{JOINCOMPLETE}, sid, jsid, tsk)$ from $\mathcal{S}$.
   - Abort if $\mathcal{I}$ or $\mathcal{M}_j$ is honest and a record $\langle \mathcal{M}_j, *, * \rangle \in$ `Edge Members` already exists.
   - If $\mathcal{M}_j$ and E_j are honest, set $tsk \leftarrow \perp$.
   - Else, verify that the provided tsk is eligible by checking
     - CheckkeyHonest$(tsk) = 1$ if $\mathcal{M}_j$ is honest and E_j is corrupt, or
     - CheckkeyCorrupt$(tsk) = 1$ if $\mathcal{M}_j$ is corrupt and E_j is honest.
   - Insert $\langle \mathcal{M}_j, E_j, tsk \rangle$ into `Edge Members` and output (JOINED, sid, $jsid$) to E_j.

IoT-JOIN

1. On input $(\text{JOIN}, sid, jsid', D_i)$ from IoT device D_i, create a join session record $\langle jsid', D_i, E_j \rangle$ for all $j \in [1, v]$ and output (JOINPROCEED, sid, $jsid'$, D_i) to E_j.
2. On input (JOINPROCEED, sid, $jsid'$) from E_j for all $j \in [1, v]$, output (JOINCOMPLETE, sid, $jsid'$) to $\mathcal{S}$.

3. On input (JOINCOMPLETE, $sid, jsid', dsk, w_{ij}$) from $\mathcal{S}$ for all $j \in [1, v]$.
   - Abort the join for (E_j, D_i), if there exists $j \in [1, v]$ where E_j or D_i is honest and a record $\langle D_i, *, * \rangle \in$ `IoT Members` already exists.
   - If E_j and D_i are honest, set $w_{ij} \leftarrow \perp$.
   - If D_i is honest, set $dsk \leftarrow \perp$.
   - Else, verify that the provided dsk is eligible by checking
     - CheckkeyHonest$(dsk) = 1$ if D_i is honest, or
     - CheckkeyCorrupt$(dsk) = 1$ if D_i is corrupt.
   - Insert $\langle D_i, E_j, dsk, w_{ij} \rangle$ into `IoT Members` for all $j \in [1, v]$ and output (JOINED, sid, $jsid'$, D_i) to E_j.

IoT-SIGN

1. For all $i \in [1, c]$, where c is the number of children for an edge device E_j. On input (SIGN, $sid, ssid, m$) from D_i. If E_j is honest and no entry $\langle D_i, E_j, w_{ij}, * \rangle$ in `IoT Members`, abort. Else, create a sign session record $\langle ssid, D_i, m \rangle$ and output (SIGNPROCEED, sid, $ssid$) to D_i.
2. On input (SIGNPROCEED, sid, $ssid$, m) from D_i for all $i \in [1, c]$. Output (SIGNCOMPLETE, sid, $ssid$) to $\mathcal{S}$.
3. On input (SIGNCOMPLETE, sid, $ssid$, σ_i,$m, \sigma_{[1-c]}$) from $\mathcal{S}$.
   - For each $i \in [1, c]$, if D_i is honest, ignore the adversary's signature σ_i and internally generate the signature for a fresh or established dsk:
     - Retrieve dsk from $\langle D_i, dsk \rangle \in$ `IoTDomainKeys` . If no such dsk exists, set $dsk \leftarrow$ Kgen().
     - Check CheckkeyHonest$(dsk) = 1$ and store $\langle D_i, dsk \rangle$ in `IoTDomainKeys`
     - Compute signature as $\sigma_i \leftarrow \text{sig}_{\text{IoT}}(dsk, m)$ and check $\text{ver}_{\text{IoT}}(\sigma_i, m) = 1$.
     - Check $\text{identify}_{\text{IoT}}(\sigma_i, m, dsk) = 1$ and check that there is no $D' \neq D_i$ with key dsk' registered in `IoT Members` or `IoTDomainKeys` with $\text{identify}_{\text{IoT}}(\sigma_i, m, dsk') = 1$.
     - Calculate $\sigma_{[1-c]} \leftarrow$ aggregate($\sigma_1, \sigma_2, \ldots, \sigma_c$) if all σ_i are honestly generated.
   - Store $\langle \sigma_i, m, D_i \rangle$ in `IoTSigned`.
   - Output (SIGNATURE, sid, $ssid$, σ_i, w_{ij}) to E_j.
   - Output (SIGNATURE, $sid, ssid, \sigma_{[1-c]}, \sigma_{i \in [1,c]}$) to E_j.

Edge SIGN

1. On input (SIGN, sid, $ssid$, $\mathcal{M}_j$, bsn) from host E_j. If $\mathcal{I}$ is honest and no entry $\langle \mathcal{M}_j, E_j, * \rangle$ exists in `Edge Members`, abort. Else, Create a sign session record $\langle ssid, \mathcal{M}_j, E_j, \text{bsn} \rangle$ and output (SIGNPROCEED, sid, $ssid$, bsn) to $\mathcal{M}_j$.
2. On input (SIGNPROCEED, sid, $ssid$) from $\mathcal{M}_j$. Output (SIGNCOMPLETE, sid, $ssid$) to $\mathcal{S}$.

3. On input (SIGNCOMPLETE, sid, $ssid$, $\sigma_{DAA}, C, \sigma_r$) from $\mathcal{S}$.

   - If $\mathcal{M}_j$ and E_j are honest, ignore the adversary's signature and internally generate the signature for a fresh or established tsk, and a fresh ring signing key r:
     - If bsn $\neq \perp$, retrieve tsk from $\langle \mathcal{M}_j, \text{bsn}, tsk \rangle \in$ `EdgeDomainKeys` for $(\mathcal{M}_j, \text{bsn})$. If no such tsk exists or bsn $= \perp$, set $tsk \leftarrow$ Kgen(). Check CheckkeyHonest$(tsk) = 1$ and store $\langle \mathcal{M}_j$, bsn, $tsk \rangle$ in `EdgeDomainKeys`.
     - $C \leftarrow_{Randomly}$ `configurationList` (Configuration Privacy) (based on the hiding of the commitment scheme) then compute the DAA and ring signatures by running the signing algorithms $\sigma_{DAA} \leftarrow \text{sig}_{\text{DAA}}(tsk, C, \text{bsn})$ and $\sigma_r \leftarrow \text{sig}_{\text{ring}}(r, C)$.
     - Check $\text{ver}_{\text{DAA}}(\sigma_{DAA}, C, \text{bsn}) = 1$ and $\text{ver}_{\text{ring}}(\sigma_r, C)$=1 .
     - Check $\text{identify}_{\text{DAA}}(\sigma_{DAA}, C, \text{bsn}, tsk) = 1$ and check that there is no $\mathcal{M}' \neq \mathcal{M}_j$ with key tsk' registered in `Edge Members` or `EdgeDomainKeys` with $\text{identify}_{\text{DAA}}(\sigma_{DAA}, C, \text{bsn}, tsk') = 1$.
   - If E_j and $\mathcal{M}_j$ are honest, store $\langle \sigma_{DAA}, C, \sigma_r$, bsn, $\mathcal{M}_j$, $E_j \rangle$ in `EdgeSigned`.
   - Output (SIGNATURE, sid, $ssid$, σ_{DAA}, σ_r) to E_j.

DAA-VERIFY: On input (VERIFY, sid, C, σ_r, bsn, σ_{DAA}, `KRL`) from some party $\mathcal{V}$.

- Retrieve all pairs $(tsk_j, \mathcal{M}_j)$ from $\langle \mathcal{M}_j, *, tsk_j \rangle \in$ `EdgeDomainKeys` where $\text{identify}_{\text{DAA}}(\sigma_{DAA}$, C, bsn, $tsk_j) = 1$. Set $f = 0$ if at least one of the following conditions hold:

CHECK1. $C \notin$ `configurationList` (Evidence authentication)(based on the binding of the commitment scheme)

CHECK2. More than one key tsk_j was found.

CHECK3. $\mathcal{I}$ is honest and no pair $(tsk_j, \mathcal{M}_j)$ was found.

CHECK4. There is an honest $\mathcal{M}_j$ but no entry $\langle *, C, \sigma_r, bsn, \mathcal{M}_j \rangle \in$ `EdgeSigned` exists.

CHECK5. There is a $tsk^* \in$ `KRL` where $\text{identify}_{\text{DAA}}(\sigma_{DAA}, C, bsn, tsk^*) = 1$ and no pair $(tsk_j, \mathcal{M}_j)$ for an honest $\mathcal{M}_j$ was found.

- If $f \neq 0$, set $f = \text{ver}_{\text{DAA}}(\sigma_{DAA}, C, bsn)$ and $\text{ver}_{\text{ring}}(\sigma_r, C)$.
- Add $\langle \sigma_{DAA}, C, \sigma_r, bsn, \text{KRL}, f \rangle$ to `VerResults`, output (VERIFIED, sid, f) to $\mathcal{V}$.

IoT-VERIFY/ Trace: On input (VERIFY, sid, m, σ_i, `IoT-KRL`) from some party $\mathcal{V}$.

- Retrieve all pairs (dsk_i, D_i, w_{ij}) from `IoT Members` and $\langle \mathcal{D}_i, *, dsk_i \rangle \in$ `IoTDomainKeys` where $\text{identify}_{\text{IoT}}(\sigma_i, m, dsk_i) = 1$. Set $f = 0$ if at least one of the following conditions hold:

CHECK1. More than one key dsk_i was found.

CHECK2. If the parent edge E_j is honest and no pair (dsk_i, D_i, w_{ij}) was found.

CHECK3. There is an honest D_i but no entry $\langle *, m, D_i, w_{ij} \rangle \in$ `IoTSigned` exists.

CHECK4. There is a $dsk^* \in$ `IoT-KRL` where $\text{identify}_{\text{IoT}}(\sigma_i, m, dsk^*) = 1$ and no pair (dsk_i, D_i) for an honest D_i was found.

- If $f \neq 0$, set $f = \text{ver}_{\text{IoT}}(\sigma_i, m)$.

- Add $\langle \sigma_i, m, \texttt{IoT-KRL}, f \rangle$ to `IoTVerResults`, output (VERIFIED, sid, f) to $\mathcal{V}$.

DAA-LINK: On input (LINK, sid, σ_{DAA}, σ'_{DAA}, C', C, bsn) from some party $\mathcal{V}$ with bsn $\neq \perp$.

- Output $\perp$ to $\mathcal{V}$ if at least one signature tuple (σ, bsn) or (σ', bsn) is not valid (verified via the `verify` interface with $\texttt{KRL} = \emptyset$).
- For each tsk_j in and `EdgeDomainKeys` compute $b_j = \text{identify}_{\text{DAA}}(\sigma, \text{bsn}, tsk_j)$ and $b'_j = \text{identify}_{\text{DAA}}(\sigma', \text{bsn}, tsk_j)$ and do the following:
  - Set $f = 0$ if $b_j \neq b'_j$ for some j.
  - Set $f = 1$ if $b_j = b'_j = 1$ for some j.
- If f is not defined yet, set $f = \text{link}(\sigma, \sigma', \text{bsn})$.
- Output (LINK, sid, f) to $\mathcal{V}$.

5.2 Swarm Attestation Protocol Security Proof

As we have previously defined the UC security model of the ASSURED Swarm Attestation protocol, we can now proceed with the security proof of the designed scheme, leveraging the **Discrete Logarithm (DL)** and **Decisional Diffie-Hellman (DDH)** assumptions. This can be expressed as follows:

Definition 1. *(The Discrete Logarithm (DL) assumption [45]) Given $y \in G$, find an integer x such that $g^x = y$.*

Definition 2. *(Decisional Diffie-Hellman (DDH) assumption [7]) Given g^a and g^b, for random and independent integers $a, b \in \mathbb{Z}_q$, a computationally bounded adversary cannot distinguish g^{ab} from any random element g^r in the group.*

Based on the above definitions, the security of the Swarm Attestation scheme is proved using the UC model. The method of reduction to absurdity is used to prove the scheme's security. Since the security of our scheme depends on the Elliptic Curve Decisional Diffie–Hellman problem, reducing the hardness of the ECDDH assumption is made.

High level description of the security proof

We start with the real world protocol execution in Game 1. In the next game, we construct one entity C that runs the real world protocol for all honest parties. Then we split C into two pieces, an ideal functionality $\mathcal{F}$ and a simulator $\mathcal{S}$ that simulates the real world parties. Initially, we start with an "empty" functionality $\mathcal{F}$. With each game, we gradually change $\mathcal{F}$ and update $\mathcal{S}$ accordingly, moving from the real world to the ideal world, and culminating into the full protocol $\mathcal{F}$ being realized as part of the ideal world, thus, proving our proposed security model presented in Section 5.1.3. The endmost goal of our proof is to prove the indistinguishability between Game 1 and Game 16, i.e., between the complete real world and the fully functional ideal world. This is done by proving that each game is indistinguishable from the previous one starting from Game 1 to reach Game 14. As aforementioned, our proof starts with setting up the real world games (Game 1 and Game 2), followed by introducing the ideal functionality in Game 3. At this stage

the ideal functionality $\mathcal{F}$ only forwards its inputs to the simulator who simulates the real world. From Game 4 onward, $\mathcal{F}$ starts executing the setup interface on behalf of the Issuer. Moving on to Game 5, $\mathcal{F}$ handles simple $VERIFICATION$ and $LINKING$ checks without performing any detailed checks at this stage; i.e., it only checks if the device belongs to a revocation list separately. In Games 6-7, $\mathcal{F}$ executes the $JOIN$ interface while performing checks to keep the consistency of registered keys. It also adds checks that allow only the devices that have successfully been enrolled to create signatures. Games 8-9 proves the anonymity, both for id and the evidence, of the protocol by letting $\mathcal{F}$ handle the sign queries on behalf of honest devices through creating Swarm signatures using freshly generated random keys instead of running the sign algorithm using the device's signing key. At the end of this game we prove that, relying on DDH and DL constructions, an external environment will notice no change from previous games where the real world sign algorithm was executed. Now moving to Games 10 - 15, we let $\mathcal{F}$ to perform all other checks that are explained in our UC model (§ 5.1.3).

We use the "$\approx$" sign to express games' indistinguishability. $\mathcal{F}$ and $\mathcal{S}$. Each of these games contains further checks that guarantee a desired security requirement, while proving game to game indistinguishability. At the end of the proof, we showcase that our real protocol guarantees the desired security requirements that the ideal world provides.

Proof. **Game 1 (SWARM Real-World)**: This is the real world protocol.

Game 2 (Transition to the Ideal World): An entity C is introduced. C receives all inputs from the honest parties and simulates the real world protocol for them. This is equivalent to Game 1, as this change is invisible to $\mathcal{E}$.

Game 3 (Transition to the Ideal World with Different Structure): We now split C into two parts, $\mathcal{F}$ and $\mathcal{S}$, where $\mathcal{F}$ behaves as an ideal functionality. It receives all the inputs and forwards them to $\mathcal{S}$, who simulates the real world protocol for honest parties and sends the outputs to $\mathcal{F}$. $\mathcal{F}$ then forwards these outputs to $\mathcal{E}$. This game is essentially equivalent Game 2 with a different structure.

Game 4 ($\mathcal{F}$ handles the setup): $\mathcal{F}$ now behaves differently in the setup interface, as it stores the algorithms defined in Section 5.1.2. $\mathcal{F}$ also performs checks and ensures that the structure of sid, which represents the issuer's unique session identifier defined in Section 5.1.3, is correct for an honest $\mathcal{I}$, and aborts if not. When $\mathcal{I}$ is honest, $\mathcal{S}$ will start simulating it. Since $\mathcal{S}$ is now running the Issuer, it knows its secret key. In case $\mathcal{I}$ is corrupt, $\mathcal{S}$ extracts $\mathcal{I}$'s secret key from π_{ipk} and proceeds to the setup interface on behalf of $\mathcal{I}$. By the simulation soundness of π_{ipk}, this game transition is indistinguishable for the adversary. Game 4 $\approx$ Game 3.

Game 5 ($\mathcal{F}$ handles the verification and linking): $\mathcal{F}$ now performs the verification and linking checks instead of forwarding them to $\mathcal{S}$. There are no protocol messages and the outputs are exactly as in the real world protocol. However, the only difference is that the verification algorithms that $\mathcal{F}$ uses (namely ver_{DAA} and ver_{IoT}) do not contain revocation checks, so knowing `KRL` and `IoT-KRL` for corrupt edge and IoT respectively, $\mathcal{F}$ can perform these checks separately and the outcomes are equal, Game 5 $\approx$ Game 4.

Game 6 ($\mathcal{F}$ handles the join/ Edge-IoT authentication Channel (Dynamic Topology)): The join interface of $\mathcal{F}$ is now changed. Specifically, $\mathcal{F}$ stores the members that joined in its records. If $\mathcal{I}$ is honest, then $\mathcal{F}$ stores the secret keys tsk and x_L and w_{ij} extracted from π^1 and π_L by $\mathcal{S}$ for corrupt edge and/or IoT devices, respectively. $\mathcal{F}$ sets the tracing key for each honest edge

device (as it already knows its key tsk) or calculates it from the extracted tsk (for corrupt edge devices).Only in the case where the edge or the IoT device is already registered in `Edge Members` or `IoT Members`, $\mathcal{F}$ will abort the protocol. However, $\mathcal{I}$ and the parent edge device E_j have already tested this case before continuing with the query JOINPROCEED, thus, $\mathcal{F}$ will not abort. Knowing tsk and all children dsks, $\mathcal{F}$ now performs the IoT-Edge join.

If E_j and D_i are honest, $\mathcal{F}$ safely chooses a random masking term r and calculates $R_{ij} = h_j^r$ of k_{ij} and adds $\langle D_i, E_j, \textit{dsk}, R_{ij}\rangle$ into `IoT Members`. If E_j or D_i is corrupt, $\mathcal{F}$ knows w_{ij} extracted by the simulator. We argue that no external environment can distinguish R_{ij} for honest devices from w_{ij}. Suppose that an environment can distinguish between $w_{ij} = h_j^{k_i k_j}$ from $R_{ij} = h_j^r$. This breaks the Decisional Diffie–Hellman DDH problem. Therefore, $\mathcal{F}$ and $\mathcal{S}$ can interact to simulate the real world protocol in all cases. Due to the simulation soundness of π^1 and π_L and the hardness of the Decisional Diffie-Hellman (DDH) problem, Game 6 $\approx$ Game 5.

Game 7 (Further Join-Checks): If $\mathcal{I}$ is honest, then $\mathcal{F}$ only allows the devices that joined to sign. An honest edge device will always check whether it joined with a TC in the real world protocol, so there is no difference for honest edge devices. In the case that an honest $\mathcal{M}_j$ performs a join protocol with a corrupt edge device E_j and an honest Issuer, $\mathcal{S}$ will make a join query with $\mathcal{F}$, to ensure that $\mathcal{M}_j$ and E_j are in `Edge Members`. Also, only joined IoT devices with certified public keys can create signatures. This will be checked by the parent edge device before verifying its children IoT signatures. Therefore Game 7 $\approx$ Game 6.

Game 8 ($\mathcal{F}$ handles the sign/ Anonymity) (Simulating an edge device without knowing its secret): In this game, $\mathcal{F}$ creates anonymous signatures for honest edge devices by running the algorithms defined in the setup interface. Let us start by defining Game $7.t.t'$. In this game, $\mathcal{F}$ handles the first t' signing inputs of $\mathcal{M}_t$ for some integer t, and subsequent inputs are then forwarded to $\mathcal{S}$. For $j < t$, $\mathcal{F}$ handles all the signing queries with $\mathcal{M}_j$ using algorithms defined in Section 5.1.2. For $j > t$, $\mathcal{F}$ forwards all signing queries with $\mathcal{M}_j$ to $\mathcal{S}$, who creates signatures as before. Next, from the definition of Game $7.t.t'$, we note that Game 7.0.0=Game 6. For increasing t', Game $7.t.t'$ will eventually become equal to Game $7.t + 1.0$. This is because there can only be a polynomial number of signing queries to be processed. Therefore, for large enough t and t', $\mathcal{F}$ handles all the signing queries of all TC's, and Game 7 is indistinguishable from Game $7.t.t'$.

Next, we want to prove that Game $7.t.t' + 1$ is indistinguishable from Game $7.t.t'$ (i.e. we want to prove that an external environment cannot distinguish when $\mathcal{F}$ internally handles the signing queries instead of merely forwarding them to $\mathcal{S}$ for a TC $\mathcal{M}_t$). Suppose an environment can distinguish a signature (created by an honest edge device with a secret signing key tsk) from a signature constructed by the same party but with a randomly chosen fresh tsk. Then we can use that environment to break a Decisional Diffie–Hellman DDH instance α, β, γ by simulating the JOIN protocol: The first signature can be simulated using the unknown $\log_{g_1}(\alpha)$ as tsk while for the second signature we can use the unknown $\log_\beta(\gamma)$ as tsk. In the reduction, we have to be able to simulate the TC without knowing the real TC's tsk, but merely based on $\alpha = g_1^{tsk}$. A TC uses tsk to set $Q \leftarrow g_1^{tsk}$ in the JOIN protocol; to do proofs π_1 in joining and π in signing; and to compute pseudonyms. In the simulation, we set $Q \leftarrow \alpha$ and we simulate all proofs π_1 and π. For pseudonyms, the power over the random oracle is used: $\mathcal{S}$ chooses $H_1(\mathsf{bsn}) = g_1^r$ for $r \leftarrow \mathbb{Z}_q$, and sets $nym \leftarrow \alpha^r = H_1(\mathsf{bsn})^{tsk}$ without knowing tsk. Anonymity of the IoT devices is also achieved against external Verifiers through the use of *fully random pseudonyms* that don't reveal the identities of the IoT devices. Thus, starting with an IoT pseudonym, $\mathcal{V}$ cannot tell the identity of the IoT device that generated this pseudonym unless it collaborates with the edge device. It is easy to see that an external environment cannot distinguish between an honest IoT signature

from a signature, by the same party, but with a randomly chosen fresh dsk in every signature (Game 8 $\approx$ Game 7, except for a negligible probability).

Game 9 ($\mathcal{F}$ handles the sign/Evidence Privacy): $\mathcal{F}$ further manage the sign queries. In this game, $\mathcal{F}$ provides a signature by simulating the Edge devices without knowing their configuration. If the edge device is honest, $\mathcal{F}$ samples a random $C \leftarrow_{Randomly}$ `configurationList` (any random element from the masked correct configurations) and creates signature on C without knowing the the devices secrets as in the previous game. If the Edge device is corrupt, $\mathcal{F}$ knows the value of C extracted by the the simulator where $C = g_1^{CS_E} g_2^r$ for some random $r \in \mathbb{Z}_q$. An environment $\mathcal{E}$ cannot distinguish C for honest devices provided by $\mathcal{F}$ from the C extracted from the simulator. This is achieved in real world protocol due to the perfect hiding of the commitment scheme used in [16].

$\mathcal{F}$ creates a ring signature on behalf of honest Edge devices. he ring signature is created using a uniformly random r and C. The ring signature leaks no information about C, other that $C \in$ `configurationList`, $\mathcal{F}$ creates the ring signature using the algorithm $\mathtt{sig}_{\mathtt{ring}}(r, C)$.

If the edge device is corrupt, then $CS_E \notin$ `configurationList`. $\mathcal{S}$ proceeds with creating a ring signature on CS_E as in the real world protocol. The only thing that $\mathcal{E}$ learns is that, for honest edge devices, its configuration belongs to a dummy list `configurationList` without knowing the exact value of the real configuration since C is a masking term of the real configuration. For corrupt devices, $\mathcal{E}$ can notice that CS_E is not in the correct configuration, which is perfectly deduced in the real-world scenario due to the privacy of the ring signature in [16]. (Game 9 $\approx$ Game 8, except for a negligible probability).

Game 10 (Traceability): When storing a new tsk or dsk, $\mathcal{F}$ checks if $\mathsf{CheckkeyHonest} = 1$ or $\mathsf{CheckkeyCorrupt} = 1$ for both keys. If the device is corrupt, $\mathcal{F}$ checks that $\mathsf{CheckkeyCorrupt} = 1$ for the keys tsk or/and dsk that the simulator extracted. This check prevents the adversary from choosing different keys. There exists only a single tsk for every valid signature where $\mathsf{identify}_{\mathsf{DAA}}(\sigma, \mu, bsn, tsk) = 1$, and only a single dsk for every valid signature where $\mathsf{identify}_{\mathsf{IoT}}(\sigma_i, m, dsk) = 1$ for each edge and IoT device, respectively, thus this check will never fail. For keys of honest devices, $\mathcal{F}$ verifies that $\mathsf{CheckkeyHonest} = 1$ whenever it receives or generates a new key. With these checks, we avoid the registration of keys for which matching signatures already exist. Since keys for honest devices are chosen uniformly from an exponentially large group and every signature has exactly one matching key, the chance that a signature under that key already exists is negligible, Game 10 $\approx$ Game 9.

Game 11 (Correctness): In this game, $\mathcal{F}$ checks that honestly generated signatures are always valid. This is true, since the $\mathsf{sig}_{\mathsf{DAA}}$, $\mathsf{sig}_{\mathsf{ring}}$ and $\mathsf{sign}_{\mathsf{IoT}}$ algorithms always produce signatures passing through verification checks. Also, they satisfy $\mathsf{identify}_{\mathsf{DAA}}(tsk, \sigma_{DAA}, \mu, bsn)=1$ and $\mathsf{identify}_{\mathsf{IoT}}(dsk, \sigma_i, m)=1$. $\mathcal{F}$ ensures, by using its internal records MemberList and DomainKeys, that honest users are not sharing the same secret key tsk or dsk; this is reduced with non-negligible probability to solving the DL problem. Assume that $\mathcal{F}$ receives an instance $h \in G_1$ of the DL problem and must answer $log_{g_1}(h)$. $\mathcal{F}$ chooses an honest device and simulates its tasks using the unknown discrete logarithm of h as its secret key. When a tsk/dsk matches one of this device's signatures in the revocation list, then this must be the discrete log of h, as there is only one tsk/dsk matching a signature.

Also, due to the binding property of the commitment scheme that binds the configuration to the commitment produced and the correctness of the ring signature scheme used, a ring signature

only verifies if the configuration belongs to the correct configuration list in the real world with is achieved in the ideal world for honest devices since C is sampled from a pre-defined dummy list of configuration `ConfigurationList`. Therefore, Game 11 $\approx$ Game 10.

Game 12 (Non-frameability): CHECK1 ensures that there are no multiple tsk or dsk values matching one signature. $\mathcal{F}$ also checks, with the help of its internal key records Members and DomainKeys, that no other device already has a key which would match this newly generated signature. If this fails, we can solve the DL problem: We simulate a TC using the unknown discrete logarithm of the DL instance as tsk or dsk as previously described in the DDH reduction. If a matching tsk or dsk is found, then we have a solution to the DL problem. Therefore, as long as solving the DL problem is computationally infeasible, Game 12 $\approx$ Game 11.

Game 13 (Unforgeability): CHECK3 is added to $\mathcal{F}$ to prevent anyone from forging signatures by using honest tsk or dsk and their credentials. If the issuer is honest, CHECK2 prevents signing with join credentials that were not issued by the issuer. The unforgeability of our protocol is built on the unforgeability of the CL [11], combined with the unforgeability of the aggregate signatures [8]. The unforgeability of the Property-based Attestation (i.e the ring signature combined with the commitment scheme) is based on the unforgeability of the scheme in [16]. Game 13 $\approx$ Game 14.

Game 14 (Revocation) CHECK4 is added to $\mathcal{F}$. This ensures that honest devices are not being revoked. If an honest device is simulated, when a matching key in the revocation list is found, it must be the secret key of the target instance. This is again equivalent to solving the DL of the problem, Game 14 $\approx$ Game 13.

Game 15 (Linkability): All the remaining checks of the ideal functionality $\mathcal{F}$ that are related to link queries are now included. Considering the fact that if a tsk and dsk only match one signature and no other signature, Game 15 is indistinguishable from Game 14, and $\mathcal{F}$ now includes all the functionalities of $\mathcal{F}$. This concludes the proof.

□

Chapter 6

Implementation and Performance Evaluation of ASSURED SA

6.1 Instantiation in the context of the Use Cases

The SA scheme that has been described and analyzed throughout this deliverable has been designed in order to be applied in large-scale "Systems-of-Systems" (SoS) and service graph chains, comprised of devices with varying security and privacy requirements and running mixed-criticality services. These are represented in ASSURED by the envisioned use case demonstrators, who capture a wide range different industrial and application domains. Here, we position the presented swarm attestation scheme in some example implementations of the use cases, and we demonstrate how it can be used in order fulfill the relevant security and privacy requirements.

First of all, as elaborated in Chapter 4, in the case of the "*Smart Satellites*" scenario, where multiple deployed satellites (operating in Low Earth Orbit (LEO)) need to exchange data with a Ground Station (GS) as part of the loaded (safety-critical or monitoring) application, there is a need for strong **integrity guarantees on the correctness of each Ground Station**, participating in the constructed data path in each communication session, and needs to be executed in a **timely manner** so that it does not strip any of the time available (during the communication window) for the transmission of the mission-related data.

Furthermore, Consider, for instance, the case of the "*Smart Manufacturing*" scenario, where multiple Robot Program Logic Controllers (PLCs), as part of the Real-time Monitoring System (RTM), need to be able to provide (real-time) information on the location of a robot arm to the IoT Gateway as the data controller for checking whether the robot is getting in close proximity to any of the workers, moving around on the manufacturing floor, in order to see if it needs to instruct the robot to stop its movement ($ShutDown$). As can be understood, this is a rather **safety-critical decision** that needs to be calculated, in real-time, while of course having adequate **integrity guarantees on the correctness of the monitored location data**. This translates to either having a concrete identification of the software resources to be attested periodically (only the small codebase which is responsible for the location monitoring), so that an appropriate performance balance can be achieved, or to be able to identify which subset of these PLCs need to be **attested in specific time instances based on their attestation history as recorded on the ASSURED Blockchain infrastructure**. Another option would be to schedule and orchestrate the execution of different attestation schemes for specific sets of devices depending on their exact functionality in the context of the overall service graph chain: For instance, some PLCs might need to attest to the correct execution of the vertical movement function, of a robot, while other might need to

attest to the correct update of the data variable that controls the speed of the robot. Thus, been able to attest at the same time different system properties for the same swarm of devices.

On the other hand, consider the example in the "*Smart Aerospace*" use case where all on-board Electronic Control Units (ECUs) need to be able to provide operational information (collected during the duration of a flight) to the Ground Station (through the Secure Server Router (SSR) that can be acting as the Prover) accompanied with guarantees on the correctness of this data; *no ECU was compromised that may have altered the output of such operational data*. Thus, in such a flow, it is more **important to have strong assurance claims on the correct configuration and execution of all ECUs since there are no immediate time constraints** on the performance and collection of the required information when the airplane is on the ground. Thus, in this scenario, different ECUs will need to provide evidence based on the execution of either Configuration Integrity Verification (CIV) or Control-Flow Attestation (CFA) tasks but in a privacy-preserving manner so that no unauthorized entity or stakeholder, acting as Verifier, can fingerprint the type of aircraft based on the attested software configuration profile. This necessitates for the second feature implemented towards **attestation evidence privacy**.

In addition, the privacy-preserving capabilities provided by the ASSURED SA scheme is particularly important in the scenario of "*Smart Cities*", where a variety of devices (such as cameras and smoke sensors) collect data towards achieving public safety. In this case, we aim to simultaneously attest to the correctness of the operational state of multiple such devices simultaneously while protecting their **identity privacy**, since the collected data may be personally identifiable and may contain privacy-sensitive information. This means that we need to ensure that no such identifiable information will be leaked during the execution of the swarm attestation protocol unless an authorized entity makes a request: In the case (for instance) of police enforcement bodies, it should be possible to have access to the necessary link tokens for associating an (attestation) result back to the data source - especially if we are referring to deployed video cameras where correctness of configuration software profiles need to be attached (as security claims) to any video stream provided and for which such authenticated users should be able to have full linkability and identification of the respective data sources.

In the following, we provide a **detailed experimental evaluation and benchmarking for the multiple phases of the newly designed swarm attestation scheme**. By focusing on the resources and time requirements of each operation taking place (both inside the TC and as part of the host), we showcase the efficiency of such a scheme (despite the advanced crypto primitives employed) to be deployed in resource-constrained ecosystems as the ones encountered in the aforementioned use cases.

6.2 Evaluation Methodology

We analytically evaluate the computational complexity and performance of our protocol by measuring the execution time of the core phases described in Chapter 4. These can be divided into (i) offline, i.e., the operations which can be pre-computed or do not need to be executed in real-time (such as the $SETUP$ phase described in Section 4.3.1) and (ii) online operations. Since we are interested in the evaluation of the protocol during runtime, the experiments presented here focus the online operations (described in Section 4.3.2 and Section 4.3.3). We also provide experimental results on the offline operations in Section 6.4.

Here, we present experimental results that serve to evaluate the performance of the ASSURED Swarm Attestation scheme in terms of the **execution time** and **overhead** induced bu the core

phases of the scheme, as described in Chapter 4. In the following, we divide the evaluated operations into two classes:

With regards to the experimental setup, we aimed to utilize a testbed and parameters that enabled us to better capture a real-world resource-constrained environment. Specifically, each IoT device was simulated through a **Raspberry Pi 3 (ARM v7) microcontroller**, while we aimed to approach Edge devices as nodes with a larger computational power and storage capacity. Specifically, the results were generated using a laptop with a **Intel(R) Core(TM) i7-8665U CPU 1.90-2.11GHz** processor. In addition, we used a TPM as the underlying root-of-trust in each edge device as a basis for the creation of the aforementioned crypto primitives and theis secure storage. We also employed a DAA scheme [44] enhanced with traceability and linkability features, which constitutes a novelty of ASSURED, as it is the first project of its kind to offer a **complete instantiation of such a strong and provable privacy-preserving swarm attestation mechanism**.

6.3 Evaluation of Online Operations

6.3.1 Signature Construction

The first aspect of the ASSURED Swarm Attestation scheme that we will evaluate is the **construction of signatures by IoT devices**. Recall that, from the side of the IoT devices, performing the ASSURED SA scheme consists of two sequencial steps: (i) hash computation for the generation of attestation evidence, and (ii) signature computation. The results for the hashing operation are presented in Table 6.1.

Code length (bytes)	SHA-256	Blake2s
32	0.052543ms	0.043423ms
256	0.047768ms	0.031259ms
4096	0.144122ms	0.122362ms

Table 6.1: Timing results of hashing in (ms)

From these results, we observe that the computation cost of the respective hash operations is efficient even for a rather large codebase ($\approx 4KB$) integrity check (note thatin such IoT actuators, the average code size is $\approx 2KB$). Similarly, the time required to perform signature generation, as it was described in Section 4.3.2.1, was measured to be $\approx 0.7ms$. It is evident that the online operations, performed by the IoT device, induce significantly lower overhead on the overall ASSURED SA protocol compared to the tasks of the Attestation phase executed on the edge device, including verification of IoT device signatures, construction of aggregate attestation result signature (Section 4.3.3), and creation of an anonymous (but traceable) DAA signature on the aggregate attestation result (Section 4.3.2.2) for the Verifier to verify the attestation report and to trigger the resolution of a failed attestation to the potentially compromised device that caused the failure.

6.3.2 Edge Device Aggregation and Verification

The time required for the execution of the aggregation operation (of IoT signed attestation reports), from the edge device, and the subsequent aggregate attestation result verification conducted by the Verifier, for the target swarm, is summarized in Table 6.2 for varying numbers of IoT signatures. From these results, we observe that the **complexity of the signature aggregation**

(leveraging bilinear aggregate signatures) is close to **linear to the number of signatures that should be aggregated by an Edge Device**, and actually have a linear correlation coefficient of 0.9863. In Figure 6.1, we provide a visual representation of these results. This observation is significant, since it indicates that the complexity does not increase (e.g., by growing exponentially) with the number of signatures, thus making the aggregation operation efficient for a large number of signatures.

No. of IoT signatures	Aggregation	Verification
50	0.6ms	207ms
100	0.8ms	317.2ms
300	1.7ms	753ms
500	2.5ms	1255.1ms
1000	3.3ms	2219.8ms
5000	19.8ms	9086.6ms
10000	39.8ms	17889.6ms
20000	105.3ms	57057.1ms
50000	311.8ms	166418ms
70000	442.3ms	188652.4ms
100000	480.2ms	206185.6ms

Table 6.2: Timing results for IoT signatures in (ms)

No. of DAA signatures	Aggregation
10	0.02ms
100	0.04ms
500	0.1ms
10000	1.3ms
100000	7.2ms

Table 6.3: Timing results for DAA signatures in (ms)

The most computationally intense operation of our protocol is the (online) **verification process**. This can be attributed to the fact that it first needs to **check the validity of each Edge Device DAA signature** on the aggregate result with regards to to the basename leveraged (dictating the anonymity level), and afterwards **assert the validity of the overall aggregate result** (Section 4.3.3).

The former corresponds to the DAA VERIFY phase, experimental timing results timings (Mean and Standard Deviation) for which are provided in Table 6.4. Note that this phase is split into two operations: the **verification of the (randomized) DAA credential** which takes up $\approx 155ms$ (irrespective of whether the bsn changes for every signature or not), and the **verification of the ECDSA signature**, which takes up $\approx 20ms$. Note that, in case a basename bsn is used (columns $bsn \neq \perp$), the signature verification time is slightly increased by $5ms$ compared to the case where a basename is not used (columns $bsn = \perp$). In any case, this is a quite efficient operation, and since it takes place in the host device, the usage of more powerful processors can also reduce the execution time.

For the latter, the slow verification time (as shown in Table 6.2) can mainly be attributed to the **pairing operation**, which is very expensive w.r.t. elliptic curve point additions used for the aggregation. While the aggregation increases with the number of signatures involved in the swarm, the length of aggregated signatures in the ASSURED Swarm Attestation scheme remains the same as a signature on a single message (i.e., constant size) due to the properties of the bilinear

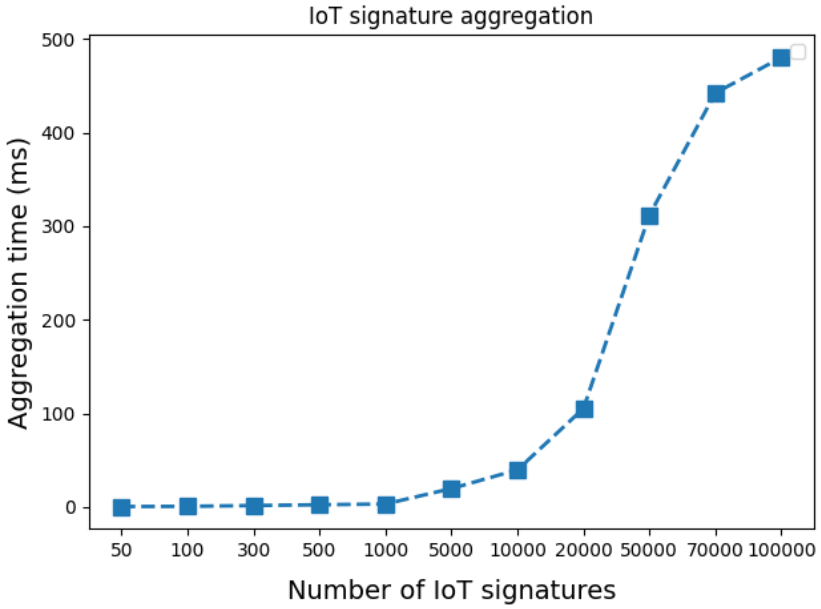

Figure 6.1: Aggregation time of a various number of IoT signatures.

aggregation signatures. Overall, this verification process is performed by the centralized Verifier, and therefore does not have significant impact on the performance of the protocol.

Time (ms)	**M**	**SD**	**M**	**SD**
Verify DAA credential	155.2	6.4	155.0	9.0
Verify signature	15.1	5.8	20.3	9.6

Table 6.4: DAA VERIFY Operation Timing

6.3.3 DAA Signature and Aggregation

In addition to the IoT signature aggregation, each Edge Device also creates a **DAA signature on the overall aggregate result** for achieving **anonymity** and **controlled linkability**. Specifically, all DAA signatures are aggregated prior to being forwarded to the root Verifier. In this context, the DAA SIGN operation takes $\approx 700ms$, as it is shown in the experimental results provided in Table 6.5.

Activity	**Mean (HW-TPM)**	$\pm$ **(95% CI)**
Total Application Stack	711.35 ms	0.57/0.55 ms
TPM2_LoadExternal	76.51	0.24/0.44
TPM2_Commit	97.69	0.03/0.05
TPM2_Hash	54.65	0.04/0.009
TPM2_Sign	57.53	0.04/0.01
TPM2_FlushContext	38.7	0.02/0.01
TPM2_LoadExternal	47.59	0.04/0.009
TPM2_NV_Read	54.2	0.02/0.009
TPM2_StartAuthSession	59.38	0.03/0.01
TPM2_PolicyNV	65.8	0.02/0.02
TPM2_Sign	84.03	0.01/0.01
TPM2_FlushContext	59.87	0.02/0.01

Table 6.5: DAA SIGN Operation Timing

From the provided results, we observe that the actual **signing operation** (performed with the TPM2_Sign command) is quite fast, and lasts $84.03ms$. However, the DAA key is protected by **key restriction usage policies**, so it can only be used if the device is in a correct state (to be able to protect against corrupt edge devices). This is instantiated by the **TPM2_NV_Read** and **TPM2_PolicyNV** commands, which verify the device state recorded in the TPM's Platform Configuration Registers (PCRs) against the policy been binded to the DAA Key by verifying both that a pseudonym has both been correctly activated, and it has not been revoked prior to being used for signing in. This additional check requires $\approx 120ms$. Also, note that the duration of $97.69ms$ for the TPM_Commit command refers to the case where linkability is not needed, therefore a basename is not used. In case linkability capabilities are needed, a basename needs to be used, which increases the execution time of this command to $224.0ms$, thus introducing a trade-off in terms of efficiency versus linkability.

Finally, we measured the **aggregation time for various numbers of 64-byte DAA signatures** (ranging from 10-100000) as presented in Table 6.3. The results demonstrate that this is a lightweight process that only slightly increases the computation cost.

Activity	**Mean (HW-TPM)**	**± (95% CI)**
Total Application Stack	605.7 ms	0.37/0.83 ms
TPM2_Create	101.82	0.01/0.01
TPM2_LoadExternal	47.24	0.04/0.01
TPM2_ActivateCredential	124.6	0.02/0.08
TPM2_Commit	67.59	0.03/0.02
TPM2_Hash	54.46	0.04/0.05
TPM2_Sign	57.04	0.03/0.01
TPM2_ActivateCredential	126.47	0.04/0.06

Table 6.6: DAA JOIN Operation Timing

Activity	**Mean (HW-TPM)**	**± (95% CI)**
Total Application Stack	226.23 ms	0.23/0.07 ms
TPM2_Commit	100.45	0.05/0.04
TPM2_Hash	54.54	0.05/0.01
TPM2_Sign	57.41	0.03/0.01

Table 6.7: DAA Key Creation Timing

6.3.4 Traceability

As it was previously mentioned, the ASSURED SA scheme provides **traceability for DAA signatures**, thus providing the capability to trace a failed aggregated attestation signature back to the Edge Device and, subsequently, to the IoT device that caused the failure. Overall, the traceability feature consists of two main computations: (i) the **generation of Traceability keys** (presented in Section 4.3.1.2), and (ii) the **traceability operation of the failed attestation** (presented in Section 4.3.3.2). In our experiments, we generate the tracing keys by considering the operations among g_2 (128 bytes) and TPM's secret key tsk (4 bytes), and the time for generating one tracing key is $\approx 0.4ms$. For the latter, we implemented the traceability feature with a 3-byte basename bsn and 4-byte TPM's secret key tsk, and the measurements show that the traceability operation takes $\approx 3ms$. The experimental results demonstrate the good performance of the ASSURED SA scheme in terms of tracing a failed attestation signature back to the source device.

6.4 Evaluation of Offline Operations

Next, we provide experimental results and performance evaluations for the offline DAA operations pertaining to the $SETUP$ phase, namely the **DAA Join** and **DAA Key creation** operations. The DAA Join operation involves the **certification and activation of the TPM attached to each edge device**, as well as the **creation of the HW-based DAA key** that is used in order to anonymize the signature created when using a pseudonym key. This operation includes the interaction with the issuer (Privacy CA) for certifying the correct creation of the ECC-based DAA key, so that the bilinear pairings with the edge devices are performed. This operation is implemented with the TPM commands shown in Table 6.6. Note that this operation is quite computationally heavy (lasting approximately $605ms$), but it is categorized as an offline operation and only needs to be performed once during the enrolment of the device. However, the creation of the DAA key itself is rather efficient, lasting approximately $226.23ms$, as shown in Table 6.7.

Chapter 7

Conclusions

In this deliverable, we provided a detailed description of the final version of the **ASSURED Swarm Attestation (SA)** scheme, which is able to fulfill the security and privacy requirements set forth by the envisioned ASSURED use case demonstrators. Specifically, we leveraged an **Enhnaced Traceable DAA** scheme that is able to provide anonymity to the swarm devices, so that no successful attestation can be linked back to the source device. However, in case of a failed attestation, it enables linkability properties in order to **trace the result to the device that caused the failure**. The proposed scheme also enables external third parties to verify the correctness of the attestation process, thus ensuring the **certifiability and accountability** of the SA scheme.

In order to construct the ASSURED SA scheme, we employed a **hierarchical system model**, where the parent Edge Devices receive signed attestation results from their children IoT Devices, and forward them to the Verifier. Afterwards, the Verifier utilises **Bilinear Aggregate Signatures** in order to aggregate the received results, which serves to reduce the length of the aggregate signature. In addition, for each device to attest to the correctness of its configuration state without revealing software and/or hardware information about itself, thus achieving evidence privacy, we employ **Property-Based Attestation (PBA)**. This is based on the use of **ring signatures** (i.e., a type of anonymous digital signatures), which are able to convince the Verifier that the device has been configured with one set of acceptable configuration specifications, without knowing the exact device configuration.

Another important feature introduced in the final version of the ASSURED SA scheme is the consideration of **dynamic network topologies**, which takes into account the mobility factor of the swarm devices, and the need to enable the verification of the IoT device outputs by any Edge device in the swarm. To this end, we introduced an updated design of the setup phase of the SA protocol (for both device and swarm initialization) to be able to consider **continuous authorization and authentication of the swarm devices**. Specifically, we introduced the ability for the Edge device to authenticate an IoT device without needing to recreate all the underlying cryptographic material, by utilizing a predefined shared secret.

In addition, we provided a detailed **security analysis**, where it was formally verified that the ASSURED SA protocol fulfills the security and privacy requirements that were envisioned during the design phase. Specifically, we employed the **Universal Composability (UC)** model, where we equated the real-world network topology to an ideal-world model (indistinguishable from the point of view of an adversary), and we added a different functionality of the scheme in each step of the analysis. This analysis also included the aforementioned usage of ring signatures, in order to formally verify the achievement of the attestation evidence privacy requirements. We also provided an **experimental analysis** of the ASSURED SA scheme, including both offline

and online operations, in order to demonstrate the effectiveness of the designed scheme, and its ability to simultaneously attest to the correctness of a large number of devices.

Based on all the above, it follows that ASSURED offers a solution that is able to fulfill all the security and privacy requirements of large-scale organizations comprising heterogeneous devices who wish to adopt the ASSURED framework. The ASSURED SA scheme also pushes the state-of-the-art in terms of the Swarm Attestation schemes proposed in the literature by including privacy-preserving capabilities, which have not yet been considered in existing schemes.

List of Abbreviations

Abbreviation	Translation
AE	Authenticated Encryption
ABE	Attribute-based Encryption
AK	Attestation Key
CA	Certification Authority
CFA	Control-flow Attestation
CIV	Configuration Integrity Verification
CSR	Certificate Signing Request
DAA	Direct Anonymous Attestation
DLT	Distributed Ledger Technology
EA	Enhanced Authorization
EK	Endorsement Key
GSS	Ground Station Server
MSPL	Medium-level Security Policy Language (MSPL)
NMS	Network Management System
Privacy CA	Privacy Certification Authority
Prv	Prover
PCR	Platform Configuration Register
PLC	Program Logic Controller
RA	Risk Assessment
RAT	Remote Attestation
SCB	Security Context Broker
SoS	Systems of Systems
SSR	Secure Server Router
S-ZTP	Secure Zero Touch provisioning
TC	Trusted Component
TLS	Transport Layer Security
TPM	Trusted Platform Module
Vf	Virtual Function
VM	Virtual Machine
Vrf	Verifier
WP	Work Package
ZTP	Zero Touch Provisioning

References

[1] Information disclosure. https://portswigger.net/web-security/information-disclosure, 2022.

[2] 5G PPP Architecture Working Group. 5G empowering vertical industries, 2016. https://5g-ppp.eu/wp-content/uploads/2016/02/BROCHURE_5PPP_BAT2_PL.pdf [Online; accessed 26-August-2017].

[3] Tigist Abera, Ferdinand Brasser, Lachlan Gunn, Patrick Jauernig, David Koisser, and Ahmad-Reza Sadeghi. Granddetauto: Detecting malicious nodes in large-scale autonomous networks. In *24th International Symposium on Research in Attacks, Intrusions and Defenses*, pages 220–234, 2021.

[4] Mahmoud Ammar, Bruno Crispo, and Gene Tsudik. Simple: A remote attestation approach for resource-constrained iot devices. In *2020 ACM/IEEE 11th International Conference on Cyber-Physical Systems (ICCPS)*, pages 247–258. IEEE, 2020.

[5] Nadarajah 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*, pages 964–975, 2015.

[6] ASSURED. Assured secure distributed ledger maintenance & data management. Deliverable D4.2, 11 2022.

[7] Dan Boneh. The decision diffie-hellman problem. In *Algorithmic Number Theory: Third International Symposiun, ANTS-III Portland, Oregon, USA, June 21–25, 1998 Proceedings*, pages 48–63. Springer, 2006.

[8] Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In *Advances in Cryptology—EUROCRYPT 2003: International Conference on the Theory and Applications of Cryptographic Techniques, Warsaw, Poland, May 4–8, 2003 Proceedings 22*, pages 416–432. Springer, 2003.

[9] Ferdinand Brasser, Brahim El Mahjoub, Ahmad-Reza Sadeghi, Christian Wachsmann, and Patrick Koeberl. TyTAN: tiny trust anchor for tiny devices. In *Proceedings of the 52nd Design Automation Conference DAC '15*, pages 1–6, 2015.

[10] J. Camenisch, L. Chen, M. Drijvers, A. Lehmann, D. Novick, and R. 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, 2017.

[11] Jan Camenisch, Manu Drijvers, and Anja Lehmann. Universally composable direct anonymous attestation. In *Public-Key Cryptography – PKC 2016*, volume 9615 of *LNCS*, pages 234–264. Springer, 2016.

[12] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. Cryptology ePrint Archive, Paper 2000/067, 2000. https://eprint.iacr.org/2000/067.

[13] Xavier Carpent, Karim Eldefrawy, Norrathep Rattanavipanon, Ahmad-Reza Sadeghi, and Gene Tsudik. Reconciling remote attestation and safety-critical operation on simple iot devices. In *Proceedings of the 55th Annual Design Automation Conference*, DAC '18, New York, NY, USA, 2018. Association for Computing Machinery.

[14] Xavier Carpent, Karim ElDefrawy, Norrathep Rattanavipanon, and Gene Tsudik. Lightweight swarm attestation: A tale of two lisa-s. ASIA CCS '17, 2017.

[15] Liqun Chen, Nada El Kassem, and Christopher J. P Newton. How to Bind a TPM's Attestation Keys with its Endorsement Key. *To appear in the Computer Journal*, 2023.

[16] Liqun Chen, Hans Löhr, Mark Manulis, and Ahmad-Reza Sadeghi. Property-based attestation without a trusted third party. In *International Conference on Information Security*, pages 31–46. Springer, 2008.

[17] The ASSURED Consortium. Assured attestation model & specification. Deliverable D3.1, 11 2021.

[18] The ASSURED Consortium. Assured layered attestation and runtime verification enablers design & implementation. Deliverable D3.2, 11 2021.

[19] The ASSURED Consortium. Operational sos process models & specification properties. Deliverable D1.3, 9 2021.

[20] The ASSURED Consortium. Risk assessment methodology & threat modelling. Deliverable D2.1, 11 2021.

[21] The ASSURED Consortium. Assured real-time monitoring and tracing functionalities. Deliverable D3.4, 2 2022.

[22] The ASSURED Consortium. Assured secure and scalable aggregate network attestation. Deliverable D3.6, 2 2022.

[23] The ASSURED Consortium. Final demonstrators implementation report. Deliverable D6.3, 8 2022.

[24] The ASSURED Consortium. Assured layered attestation and runtime verification enablers design & implementation - version 2. Deliverable D3.3, 2 2023.

[25] The ASSURED Consortium. Assured real-time monitoring and tracing functionalities - version 2. Deliverable D3.5, 2 2023.

[26] The ASSURED Consortium. Assured tc-based functionalities - version 2. Deliverable D4.6, The ASSURED Consortium, 02 2023.

[27] Heini Bergsson Debes, Edlira Dushku, Thanassis Giannetsos, and Ali Marandi. Zekra: Zero-knowledge control-flow attestation. ASIA CCS '23, page 357–371, New York, NY, USA, 2023. Association for Computing Machinery.

[28] 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*, ARES '22, New York, NY, USA, 2022. Association for Computing Machinery.

[29] Heini Bergsson Debes and Thanassis Giannetsos. Retract: Expressive designated verifier anonymous credentials. In *Proceedings of the 18th International Conference on Availability, Reliability and Security*, ARES '23, New York, NY, USA, 2023. Association for Computing Machinery.

[30] Heini Bergsson Debes, Thanassis Giannetsos, and Ioannis Krontiris. BLINDTRUST: oblivious remote attestation for secure service function chains. *CoRR*, abs/2107.05054, 2021.

[31] Danny Dolev and Andrew Yao. On the security of public key protocols. *IEEE Transactions on information theory*, 29(2):198–208, 1983.

[32] Edlira Dushku, Md Masoom Rabbani, Mauro Conti, Luigi V Mancini, and Silvio Ranise. Sara: Secure asynchronous remote attestation for iot systems. *IEEE Transactions on Information Forensics and Security*, 15:3123–3136, 2020.

[33] Karim Eldefrawy, Gene Tsudik, Aurélien Francillon, and Daniele Perito. SMART: Secure and Minimal Architecture for (Establishing Dynamic) Root of Trust. In *Proceedings of the 19th Annual Network & Distributed System Security Symposium NDSS '12*, page 1–15, 2012.

[34] Thanassis Giannetsos and Ioannis Krontiris. Securing v2x communications for the future: Can pki systems offer the answer? In *Proceedings of the 14th International Conference on Availability, Reliability and Security*, ARES '19, 2019.

[35] Stylianos Gisdakis, Marcello Lagana, Thanassis Giannetsos, and Panos Papadimitratos. SEROSA: service oriented security architecture for vehicular communications. In *VNC*. IEEE, 2013.

[36] S Goldwasser, S Micali, and C Rackoff. The knowledge complexity of interactive proof-systems. In *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing*, STOC '85, page 291–304, New York, NY, USA, 1985. Association for Computing Machinery.

[37] Ragnar Mikael Halldórsson, Edlira Dushku, and Nicola Dragoni. Arcadis: Asynchronous remote control-flow attestation of distributed iot services. *IEEE Access*, 9:144880–144894, 2021.

[38] IETF RATS Working Group. Trusted path Routing, 2023. https://www.ietf.org/archive/id/draft-voit-rats-trustworthy-path-routing-07.html#name-yang-module [Online; accessed 04-February-2023].

[39] Pooyan Jamshidi, Claus Pahl, Nabor C. Mendonça, James Lewis, and Stefan Tilkov. Microservices: The journey so far and challenges ahead. *IEEE Software*, 35(3):24–35, 2018.

[40] Patrick Koeberl, Steffen Schulz, Ahmad-Reza Sadeghi, and Vijay Varadharajan. TrustLite: A security architecture for tiny embedded devices. In *Proceedings of the 9th European Conference on Computer Systems EuroSys '14*, pages 1–14, 2014.

[41] Florian Kohnhäuser, Niklas Büscher, and Stefan Katzenbeisser. Salad: Secure and lightweight attestation of highly dynamic and disruptive networks. In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, pages 329–342, 2018.

[42] Ioannis Krontiris, Thanassis Giannetsos, Peter Schoo, and Frank Kargl. Buckle-up: autonomous vehicles could face privacy bumps in the road ahead. In *18^{th} escar Europe : The World's Leading Automotive Cyber Security Conference (Konferenzveröffentlichung)*. 2020.

[43] Benjamin Larsen, Heini Bergsson Debes, and Thanassis Giannetsos. Cloudvaults: Integrating trust extensions into system integrity verification for cloud-based environments. In *Computer Security*, pages 197–220, Cham, 2020. Springer International Publishing.

[44] Benjamin Larsen, Thanassis Giannetsos, Ioannis Krontiris, and Kenneth Goldman. Direct anonymous attestation on the road: Efficient and privacy-preserving revocation in c-its. In *Proceedings of the 14th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, WiSec '21, New York, NY, USA, 2021.

[45] Kevin S McCurley. The discrete logarithm problem. In *Proc. of Symp. in Applied Math*, volume 42, pages 49–74. USA, 1990.

[46] Wenjuan Meng, Tao Jiang, and Jianhua Ge. Dynamic Swarm Attestation With Malicious Devices Identification. *IEEE Access*, 6:50003–50013, 2018.

[47] OpenFog Consortium Architecture Working Group. OpenFog Reference Architecture for Fog Computing. https://www.iiconsortium.org/pdf/OpenFog_Reference_Architecture_2_09_17.pdf, 2017. [Online; accessed Aug 31, 2022].

[48] Sandro Pinto and Nuno Santos. Demystifying Arm TrustZone: A Comprehensive Survey. *ACM Comput. Surv.*, 51(6), 2019.

[49] Pasika Ranaweera, Anca Delia Jurcut, and Madhusanka Liyanage. Survey on multi-access edge computing security and privacy. *IEEE Comm. Surveys*, 2021.

[50] Phillip Rieger, Marco Chilese, Reham Mohamed, Markus Miettinen, Hossein Fereidooni, and Ahmad-Reza Sadeghi. Argus: Context-based detection of stealthy iot infiltration attacks. *USENIX*, 2023.

[51] Ronald L Rivest, Adi Shamir, and Yael Tauman. How to leak a secret. In *International conference on the theory and application of cryptology and information security*, pages 552–565. Springer, 2001.

[52] Stephan Wesemeyer and all. Formal analysis and implementation of a tpm 2.0-based direct anonymous attestation scheme. ASIA CCS '20, 2020.