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

ASSURED

D3.6: ASSURED SECURE AND SCALABLE AGGREGATE NETWORK ATTESATION

Revision: v.1.0

Work package	WP 3
Task	Task 3.5
Due date	28/02/2022
Deliverable lead	TUDA
Version	1.0
Authors	Richard Mitev (TUDA), Philip Rieger (TUDA)
Reviewers	Edlira Dushku (DTU) Dimitris Karras (UBITECH)
Abstract	Deliverable D3.6 designs and documents the new secure and scalable network attestation scheme (Swatm Attestation) including the necessary models, the full specification of the internal crypto primitives and amendments needed to be done to the existing ASSURED (one-to-one) attestation schemes. The ASSURED Swarm Attestation protocol is based on an enhancement of the Direct Anonymous Attestation (DAA) mechanisms, leveraged for offering different levels of privacy, while at the same time enabling selective linkability in case of a failed attestation result for one of the swarm devices.
Keywords	Swarm Attestation, DAA, Linkability, Group-based Signatures, Static Topology, Security Analysis, Linkability, Traceability

Document Revision History

Version	Date	Description of change	List of contributors
v0.1	07.12.2021	ToC	Richard Mitev (TUDA)
v0.2	24.12.2021	SOTA analysis of the variouos swarm attestation schemes based on the various features of topology, number of verifiers, etc. This sets the scene for identifying the context in which ASSURED Swamr Attestaion needs to also operate (Chapter 2)	Liqun Chen, Nada El Kassem (SURREY) Edlira Dushku, Benjamin Larsen, Heini Bergsson Debes(DTU) Richard Mitev, Philip Rieger (TUDA) Thanassis Giannetsos (UBITECH) Ilias Aliferis (UNIS)
v0.3	14.01.2022	First result of the high-level conceptual architecture of the ASSURED SA without considering linkability (Chapter 6)	Liqun Chen, Nada El Kassem (SURREY) Edlira Dushku, Benjamin Larsen, Heini Bergsson Debes(DTU) Richard Mitev, Philip Rieger (TUDA) Thanassis Giannetsos (UBITECH)
v0.4	28.01.2022	Description of the adversarial model considered for the ASSURED SA (Chapter 3)	Edlira Dushku (DTU) Dimitris Karras (UBITECH)
v0.5	11.02.2021	Description of the security, trust and privacy requirements to be achieved by the ASSURED SA (Chapter 4)	Liqun Chen, Nada El Kassem (SURREY) Edlira Dushku, Benjamin Larsen, Heini Bergsson Debes (DTU) Ilias Aliferis (UNIS) Dimitris Karras (UBITECH)
v0.6	18.02.2022	Description of the building blocks leveraged in ASSURED SA for achieving the required privacy properties (Chapter 5)	Liqun Chen, Nada El Kassem (SURREY) Edlira Dushku (DTU) Thanassis Giannetsos, Dimitris Papamartzivanos (UBITECH)
v0.7	25.02.2022	Update and finalization of the ASSURED SA considering also the design of the Enhanced and Traceable DAA (Chapter 6)	Liqun Chen, Nada El Kassem (SURREY) Edlira Dushku (DTU) Richard Mitev, Philip Rieger (TUDA) Thanassis Giannetsos (UBITECH)
v0.9	02.03.2022	Review the document	Edlira Dushku (DTU) Dimitris Karras (UBITECH)
v1.0	07.03.2022	Addition of the high-level security analysis on how the envisioned security and privacy properties are achieved (Chapter 7)	Liqun Chen, Nada El Kassem (SURREY)

Editors

Richard Mitev (TUDA), Philip Rieger (TUDA)

Contributors (ordered according to beneficiary numbers)

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

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

Liqun Chen, Nada El Kassem (SURREY)

Ahmad Atali, Meni Onrebach (MLNX)

Dimitrs Papamartzivanos, Thanassis Giannetsos, Dimitris Karras (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)

DEM: Demonstrator, pilot, prototype, plan designs

DEC: Websites, patents filing, press & media actions, videos, etc.

OTHER: Software, technical diagram, etc.

Executive Summary

In a system that is distributed to multiple autonomous devices, as so-called "*System-of-Systems*" (SoS), verifying security properties about the individual devices is currently still an open research problem. In Deliverable D3.2 [20], we introduced different attestation schemes for attesting both the behavioural and/or configuration state of individual devices. Extending such assurance claims to a network of devices, in this deliverable, we proceed in providing a **scalabe, distributed scheme that uses the aforementioned attestation enablers in a swarm context**. This deliverable, therefore, **puts forht the documentation of the newly designed secure and scalable network attestation mechanism including the necessary models and the full specification of the internal crypto primitives.**

In ASSURED, various types of attestation schemes are employed depending on the operational and assurance requirements of a specific systems including **Direct Anonymous Attestation (DAA), Configuration Integrity Verification (CIV) and Control-Flow Attestation (CFA)**. DAA is a platform authentication mechanism that enables the provision of privacy-preserving and accountable authentication services. By leveraging group-based signatures, any verifying entity can verify a platform's credentials in a privacy-preserving manner using the previously described DAA algorithms, without the need of knowing the platform's identity. The CIV validates that all the binaries loaded in the device have the expected content. That is, that no malicious adversary loaded an invalid program or changed any of the code pages of deployed programs as meant by the end-user. The control-flow-attestation inspects the state of a system at runtime, i.e., the control-flow of the executed binary to verify its integrity. We described the requirements for each scheme as well as the security properties that each attestation scheme provides, as well as the primitives.

In this document, we describe how ASSURED leverages these attestation scheme for a Secure and Scalable Aggregate Network Attestation towards establishing attestable trust to groups of devices in large-scale dynamic networks, called Swarm Attestation. We describe the required security and trustworthiness properties as well as the provided features of the first release of the ASSURED Swarm Attestation (SA) protocol based on the design of a novel and **Enhanced Traceable DAA** scheme. This enables us to both **protect the system from malicious devices**, by identifying the origin of a possible compromise, but also **protect devices from untrusted verifying entities**, especially those where the attestation history shows that they have never been compromised or rarely so, by enabling *unlinakbility*: **No successful attestation result should be linked back to its device origin as this could enable attacks including implementation disclosure attacks**. Or even enable "*honest-but-curious*" entities that do not want to impede with the protocol functionality but are interested in creating device attestation profiles - *how often a device is been required to execute an attestation task (might reveal type of safety-critical functions executed), the type of attestation task, the set of devices comprising the entire swarm, etc.*

Another aspect that we discuss is a robust strategy for handling scenarios where few devices deviate from their expected behavior, s.t., their attestation fails. More specifically, we extensively discuss the considered threat model, focusing on an adversary that can compromise devices and networks towards manipulating the correct configuration and execution of the target device, stealing confidential data and getting unauthorized access. In addition, we analyze the security properties that need to hold for the ASSURED Swarm Attestation, with special attention on the **integrity of the aggregated attestation reports, provenance of report signatures and linkability of the results for the individual devices**. The endmost goal is, based on these requirements and properties, to design the novel signature scheme to be used as part of the

ASSURED SA. To provide a detailed overview of the security context of ASSURED, we describe all the crypto primitives used/provided by ASSURED that enable the design of swarm attestation scheme and put special attention on the group signature scheme that is used for signing the attestation result. Further, we describe the exact behavior of the swarm attestation and extensively discuss the flow of the individual actions that need to be performed for the swarm attestation. At the end, we extensively discuss for each requirement that was discussed before, how the dynamic swarm attestation of ASSURED achieves it.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Towards Secure and Scalable Aggregate Attestation

Today's complex supply-chain ecosystems consist of large networks of heterogeneous devices, such as embedded and moving devices. These smart devices collaborate to provide all kinds of services to benefit a diverse set of applications, such as industry or critical infrastructures. Such networks are called swarms. However, for such distributed services to be trusted, one needs to ensure that each device in the network behaves correctly. A very effective way to ensure this is to prove the correctness of each device's software state. This can be done via remote attestation, which checks the software integrity by detecting modified execution flows or data. Unfortunately, these schemes on their own do not scale to many devices to be attested, such as a central Verifier attesting the entire collaborative network. For example, in a naive approach, individually attesting each device in the network would induce a linearly growing overhead in both communications and computations. Therefore, attesting large swarms requires a new and efficient approach.

These swarms usually consist of multiple groups of heterogeneous devices forming their own subsystems (e.g., controllers and sensors in smart industry). In these compositions of systems, so called, "Systems-of-Systems" (SoS) making valid statements about the security properties of single devices as well as groups is a key challenge ASSURED tries to resolve. This document builds on D3.2 [20], which already introduces the need of establishing **federated trust** between services and devices using decentralized solutions. Therefore, the so called Swarm Attestation schemes are a promising technology to include in the ASSURED framework, overcoming the imposed challenges.

The goal of this document is to, therefore, **present a first release of the ASSURED Secure and Scalable Aggregate Network Attestation towards establishing attestable trust to groups of devices in large-scale dynamic networks**. By attesting the correct behaviour of large groups of devices trust can be established in the whole network by forwarding proof of this validation. In order to fulfill this goal, edge devices, resource constrained devices and cloud devices need to be aware of this newly introduced overlaying attestation mechanism. The ASSURED framework has to instruct devices already supporting attestation or verification mechanisms to utilize them for attesting groups or swarms of devices.

Furthermore, the **employed swarm attestation strategy should enable the network of devices to continue to operate correctly even if some devices deviate from specified behavior, e.g., as the result of a compromise**. The mechanism should therefore be robust and correct in different environments (e.g., centric, distributed, or mobile swarms) and device configurations (e.g., different hardware capabilities). Efficiency is the main goal of this attestation strategy, en-

abling Verifiers to effortlessly validate the same attestation attribute or different ones in every Prover in an efficient way. To achieve this, the burden of attesting the whole network should be distributed equally on the network's participants. Therefore, it is possible to verify a huge swarm in a timely manner, which is crucial to reduce the time-of-check-to-time-of-use (TOCTOU) problem.

1.2 Scope and Purpose

The main purpose of this deliverable is to **present the first release of a secure and scalable aggregate network attestation mechanism for managing the trusted activities envisioned within the "*Systems-of-Systems*" ecosystem**. More specifically, it documents the Swarm Attestation mechanism, a key element of holistic SoS verification, which enables a set of entities to collectively attest their benign and correct operation towards Verifiers. In this context, D3.6 provides extensions to the attestation enablers developed in D3.2 [20] that allow the Security Context Broker to assess the trustworthiness of entire networks. Thus, the attestation burden will be distributed across the network, allowing safety-critical CPSs to attest each other and aggregate the attestation results.

In order to integrate swarm attestation into the ASSURED framework, we document the considerations and background needed to design a method suitable for ASSURED. The underlying primitives use more **efficient and lightweight crypto operations** (i.e., aggregate signatures, group-based signatures) and the high-level protocol as well as the amendments needed to fit the previously designed attestation mechanisms to work with Swarm Attestation. This allows for the efficient verification of a network, even if it consists of a huge number of devices. In addition, a new adversary model and trust assumptions need to be defined as the Swarm Attestation mechanisms imposes new requirements opposed to traditional attestation mechanisms.

1.3 Relation to other WPs and Deliverables

In what follows, Figure 1.1 depicts the relationship of this deliverable with other Work Packages (WPs) as well as the other tasks in the same WP3. As aforementioned, the main purpose of this document is to introduce a secure and scalable aggregate network attestation mechanism in the context of ASSURED. Thus, within WP3, this deliverable builds mostly on top of D3.2 [20] as all attestation schemes described herein need to be amended to be leveraged to work under the Swarm Attestation mechanism. In addition, the trust assumptions and security requirements to be achieved by the swarm attestation scheme need to be defined, thus, also building on top and amending definitions already provided in D3.1 [18] as well as D1.3 [22] and D2.1 [24]. Furthermore, while designing the Swarm Attestation protocol, previously defined policies and APIs in D2.2 [23] and D5.1 [25] need to be re-defined.

The outcome of Deliverable D3.6 is intended to support D3.7 [**?**] which designs the final Swarm Attestation protocol which is considered for security analysis. Similarly, D3.6 inspires D3.3 [**?**] which defines the appropriate attestation schemes to utilize for the proposed use cases (WP6). In relation with the rest of the WPs of the project, D3.6 adheres - through D3.2 - to the reference architecture of WP1, slightly amended Attestation Policy Models of WP2, the Blockchain infrastructure of WP4, as well as the aforementioned modified integration of WP5.

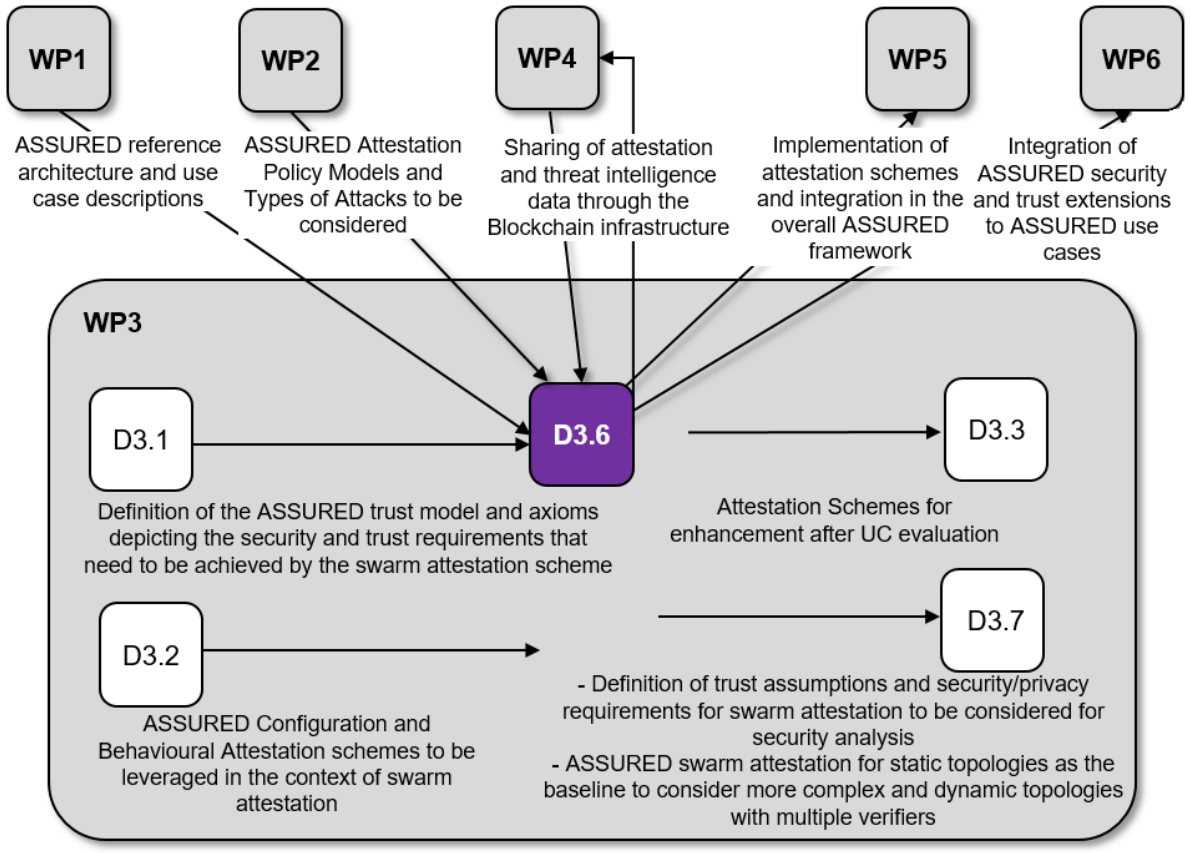

Figure 1.1: Relation of D3.6 with other WPs and Deliverables

1.4 Deliverable Structure

This deliverable is structured as follows: **Chapter 2** describes the ASSURED System Model regarding the Swarm Attestation by providing a detailed state-of-the-art analysis and specific properties that need to be considered when designing sych scalable attestation protocols. **Chapter**3 presents the adversarial model considered in ASSURED before we proceed to the detailed description of the building blocks of the attestation scheme including attestation agent and crypto primitives like the aggregate signature and group-based signatures Chapter 5. **Chapter 4** provides a description of the trust assumptions and security requirements that the ASSURED SA scheme needs to achieve for protecting against the threat model identified. In this context, we also defined the different privacy protection profiles that need to be considered depending on the operational needs of the envisioned use cases. **Chapter 6** provides detailed description of the newly designed ASSURED SA scheme listing the underpinnings of the *Traceable DAA* scheme employed towards achieving all the previously defined security and privacy properties. **Chapter 7** provides a first high-level security analysis of the described SA protocol. This is considered the first step towards the complete formal verification of the protocol against the trust model defined in D3.1 [18] to be provided by the second version of the deliverable (D3.7). Finally, **Chapter 8** concludes the deliverable.

Chapter 2

System model

2.1 ASSURED Scalable Aggregate Network Attestation

Swarm attestation schemes aim to provide **scalable attestation solutions** that **verify the trustworthiness of a large-scale network 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** in "Systems-of-Systems" comprising heterogeneous devices - with **different hardware capabilities** - loaded with (possibly) **disparate software configurations**. However, the endmost goal is not to simply reduce the communication overhead, compared to the naive approach of applying many times single Prover attestation, but to also enable **attestation intelligence on which sets of devices need to be attested collectively during run-time**. This might depend on the **safety-critical nature and time -sensitivity of the functions and services that may run in such swarms** that, in turn, may require different types of assurance claims (thus, affecting the time of attestation tasks need to be executed) to be provided based on the available resources, in a non-intrusive manner, so as to avoid harming the overall execution profile of the device.

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.

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 [20], will also enable ASSURED to make *intelligence decisions on how to adjust the execution of such security enablers so as to identify the golden balance between converging security and safety without impeding the performance of the deployed devices*. The main properties which play a crucial role in designing such an swarm attestation protocol are listed in Table 2.1 and elaborated in the subsequent sections and chapters (all security- and privacy-related requirements are descri bed in Chapter 4): **Network Topology, Number of Verifiers, Trustworthiness of Verifiers, Prover's Memory Regions considered at the Attestation, Verification of Exchanged Communication Data among Provers, Granularity of Attestation Result, Efficiency and Privacy**.

Property	Description
Network Topology	This property, as the name suggests, pertains to the topology followed by the devices in the swarm to be attested. As described in Section 2.1.1 such a topology can either be **static** or **dynamic** depending on whether devices demonstrate any **mobility** (e.g., *Smart Satellites and Smart Manufacturing use cases*) that will affect the construction of their routing trees when it comes to the parent node (with whom they communicate and might take the role of the Prover). Another aspect to take into consideration here is the need (or not) for establishing **federated trust between the devices in the swarm**. Essentially, this relates to whether a *hierarchical topology* is needed in a scenario where the device at each layer can act as the Prover for its children nodes or a *tree-based* structure is allowed where only the root node can act as the Verifier (Figure 3.2). This, of course, is directly related to the below property regarding the number of Verifiers that can be active in a swarm network. Depending on all the above, this affects the type of communication to be established per Prover device consdering also the various security and privacy needs.
Number of Verifiers	This property depicts the number of devices that can act as the Verifier in a swarm network. it can either be the case that there is only one root Verifier that is responsible for assessing the received (aggregate) result of the entire swarm; or multiple Verifiers responsible for attesting the correctness of their children nodes/devices. The latter also considers the mobility aspects that some devices might exhibit which, in turn, will lead to changing parent nodes that should be able to act as Verifiers.
Trustworthiness of Verifiers	This property reflects any trustworthiness assumptions to be made for the correctness of the Verifier device. It can either be the case that Verifiers are considered *trustworthy*, thus, they can have full knowledge of a Prover's program execution details and in-memory layout without any privacy breach (Section 4.2). This is what is the current state-of-the-art in the existing attestation landscape. However, in ASSURED, we make not assumptions on the Verifier trustworthiness been aligned with the **Zero Trust** principle that dictates that no initial trust can be assumed between entities.

Prover's System Properties for Attestation	This property pertains to the type of system properties considered for attestation. As described in D1.3 [22], these can either be execution-related (i.e., control-flow graphs) or configuration-related (i.e., hash of loaded binaries) depending on the type of attestation task employed; Control-Flow Attestation (CFA) or Configuration Integrity Verification (CIV). Another important aspect here is whether all swarm devices are attesting the same types of properties or whether different properties can be considered in the context of the samw swarm.

2.1.1 Swarm Topology

In general, swarm attestation schemes consider the attestation of two types of swarm networks: *static* or *dynamic* networks.

- **Static Swarms.** In static swarm, the network is characterized by two main properties: the network is *static* and always *interconnected*. Due to this properties, the static swarms can facilitate the construction of spanning-tree, where two connected nodes in the spanning-tree are neighbors in the communication graph. An important property of swarm attestation that rely on spanning-tree is that each device can communicate only with its direct neighbors. In these schemes, each device statically attests its children and reports back to its parent the number of children that successfully passed the attestation protocol. In the end, an aggregated report with the total number of the devices successfully attested will be transmitted to the Verifier.
- **Dynamic Swarms.** In dynamic swarms, the network topology does not remain static, it changes even while the attestation protocol is executing. Thus, the interactions in a dynamic swarm cannot be represented by a spanning-tree construction. Overall, in the dynamic attestation protocols, devices first share their respective "knowledge" with other devices, and then they use consensus mechanisms to agree on a common knowledge about the whole network.

2.1.2 Verifiers in Swarms

Based on the number of Verifiers that participate in the attestation process, swarm RA protocols can be classified into swarm protocols with *one centralized verifier* or *many distributed verifiers*.

- **One Centralized Verifier.** In a swarm RA protocol with *one centralized verifier* typically the attestation process gets initiated by a centralized Verifier. At the end of the attestation, the verifier's role is to collect the final attestation result obtained as a consequence of running the remote attestation process over a swarm. In the swarm schemes which rely on a centralized Verifier, the Verifier is considered as a *trusted* entity by design.
- **Many Distributed Verifiers.** In a swarm RA protocol with *many distributed verifiers* one Verifier can attests one or more Provers. In such schemes, the Verifier can be assumed as a *trusted* or an *untrusted* entity.

2.1.3 Prover's Attestation in Swarms

The swarm attestation design differs also from the attested memory regions of individual devices (i.e., Provers) participating in the swarm. Based on the Prover's memory regions that are con-

sidered for attestation, the swarm attestation protocols can belong to one of the following two categories: *Static Swarm Attestation* or *Control-flow Swarm Attestation*

- **Static Swarm Attestation.** In *Static Swarm Attestation* approaches, each individual device (i.e, Prover) participating in the swarm performs *static attestation*. The static attestation aims to verifies the integrity of static memory region of the Prover, and it typically consists of computing a hash of the device's memory. The main drawback of this method is that it does not apply to dynamic objects in the application. Hashing dynamic entities on a device is ineffective because they often change in an unpredictable way, making it impossible for the Verifier to keep track of their states. Runtime malware can modify dynamic objects, thus infecting the device and hiding from static attestation protocols.
- **Control-flow Swarm Attestation.** In *Control-flow Swarm Attestation* approaches, each individ a *dynamic attestation*, for instance, a control-flow attestation. Unlike performing a hash of the memory as it is the case of static attestation, the dynamic attestation schemes continuously track dynamic properties, such as the Instruction Pointer of the system, stack registers, or the application's control flow. One of the main challenges of dynamic attestation sch-emes is the difficulty in identifying the "good" dynamic properties of a system, based on which the runtime system integrity should be performed. Moreover, there is a need for a large number of dynamic entities during the attestation process, which need to be frequently measured, since their state changes all the time. However, the main advantage of dynamic attestation is its ability to detect run-time attacks, compared to static attestation which is not able to do this.

Furthermore, based on recent taxonomies of addressing security challenges and IoT attacks and the threat model considered in ASSURED (Chapter 3.1), there can also be different categories of classifying the devices of a swarm that need to be able to attest to their correct state:

- **Geographical Location:** Deployed Provers are divided into clusters based on their geographical location;
- **Software Configurations:** Deployed Provers are grouped into clusters according to the type of software they are executing and the type of OS they have loaded (partial or full similarities in their software);
- **Hardware Capabilities:** Deployed Provers are grouped into clusters according to the type of hardware they possess;
- **Security & Safety:** This is an application-dependent feature which classifies devices, D_i according to how critical they are to the application domain considering a combination of the task they perform and the security mechanisms employed by the hosting hardware;
- **Time Sensitivity:** Deployed Provers are classified into clusters according to their time constraints and how critical is it to deviate them frequently from doing their tasks.

Number of clusters in each category is an application-dependent value. Each prover should virtually belong to one cluster of each category by storing its identifier in its memory. The goal of the multi-clustering technique is twofold; First, it reduces the communication overhead in terms of number and sizes of exchanged packets. Second, over time, a Prover can learn from the attestation history the behavior of adv by realizing the clusters that have frequently compromised devices, thus being able to patch common vulnerabilities. Please note that the selection criteria of categories and clusters is flexible as more or less categories can be considered depending on the application domain.

2.1.4 Exchanged Communication Data among Provers

Another property to categorize RA protocols is based on the schemes that consider the data transferred between devices before or during the attestation process (i.e., distributed services) and those that do not (i.e., collective).

- **Collective devices.** A typical swarm attestation scheme aggregates the individual attestation results of Provers. Thus, the final attestation report represents the collective state of the Provers without considering their exchanged communication data,
- **Distributed services.** The ultimate goal of verifying the communication data exchanged between devices is not only to attest a device's trustworthiness, it also makes sure that the exchanged data was not malicious and did not maliciously affect other interacting Provers.

2.1.5 Granularity of Attestation Result

Based on the granularity of the final attestation result, swarm attestation protocols can report only the *overall* state of the network or can precisely identify compromised devices in the network. In addition, the basic swarm attestation scheme relies on a *centralized Verifier* that validates the *overall* state of the network. Such attestation outputs a Boolean result (e.g., 1 if all devices are healthy and 0 otherwise) without precisely identifying compromised devices by id.

- **Overall Network state.** The basic swarm attestation schemes relies on a *centralized Verifier* that validates the *overall* state of the network. Such attestation outputs a Boolean result, e.g., 1 if all devices are healthy and 0 otherwise.
- **Device-specific Network state.** Other attestation schemes aim to provide a comprehensive attestation report which allows the Verifiers to precisely identify the compromised devices in the network.

2.1.6 Privacy Preservation

Simple Aggregation Schemes. Conventional swarm attestation schemes adopt signature schemes which allow aggregation of each individual attestation result. In this approaches, the produced aggregated signature can be verified by any one or many verifiers. Such schemes do not deal with privacy concerns.

Privacy-Preserving Aggregation. To guarantee privacy-preserving property this, the swarm attestation will rely on the adoption of relevant cryptographic tools that allow signature aggregation while offering strong anonymity guarantees. The generation of such signatures convinces the verifier that the signature was generated by one member of the group and yet does not know who actually signed.

Traceable Privacy-preserving Aggregation. In various scenarios, the *overall* trustworthy state of the network might not be enough, especially when the network is reported to be compromised. Thus, it might be important for the verifier to identify precisely the compromised devices in the network. To enable this functionality, the signature should provide *traceability* guarantees. Generally, traceable signatures are associated with high overhead. One possible way to deal with such performance challenge it to consider a trade-off between the security, privacy and computational overhead. For instance, when there is a low risk indicator, the default swarm attestation scheme can be a privacy-preserving scheme which checks the overall network state. When there is a

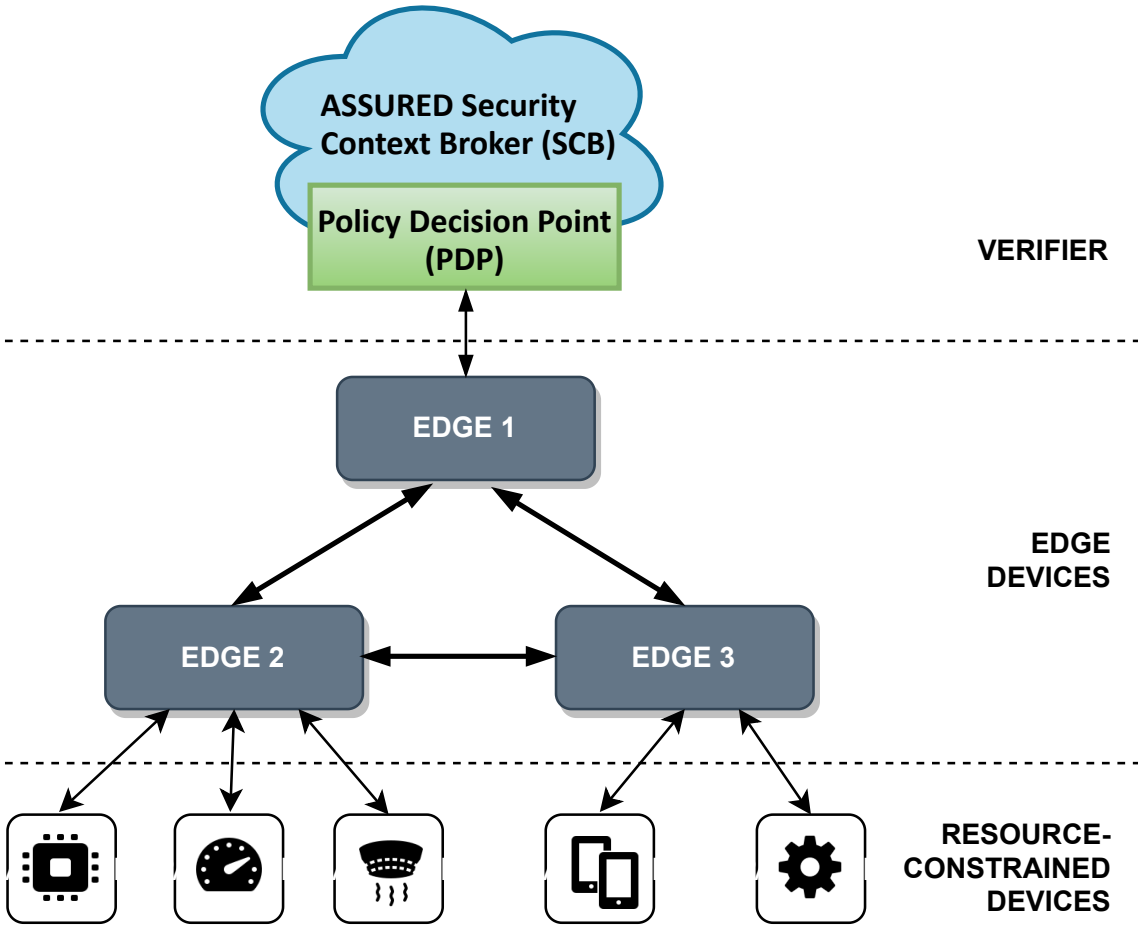

Figure 2.1: System model of Swarm attestation in ASSURED

high risk indicator or the overall network state is reported as compromised, ASSURED can run another swarm attestation approach that allows the identification of the compromised devices in a large network.

More information about the specific privacy requirements can be found in Section 4.2.

2.2 Static Swarm Attestation in ASSURED

2.2.1 System Topology for 1^{st} Release of SA

In designing a basic swarm attestation scheme, ASSURED considers *static attestation* of heterogeneous devices participating in *static network topology*, e.g., static attestation of devices deployed in smart manufacture. This network is structured as a hierarchical fog computing model as shown in Figure 3.2.

This network consists of three layers with the following characteristics:

- **Resource-constrained devices:** Each IoT device is a resource-constrained device which are not equipped with a costly specialized hardware-security modules, e.g., TPM. Consequently, it is assumed, that all the resource-constrained devices do not support the imple-

mentation of Direct Anonymous Attestation (DAA). Each IoT device is connected to only one edge device.

- **Edge devices:** Each edge device is equipped with a TPM and supports DAA. Edge devices have a parent-child relationship with the IoT devices. Thus, the edge devices are responsible for aggregating the attestation result they receive from their child IoT devices. We assume that all edge devices are in the same layer.
- **Verifier:** In ASSURED, the aggregated attestation result to be validated by the Policy Decision Point (PDP) in the ASSURED Security Context Broker (SCB).

In ASSURED, the attestation starts at the level of IoT devices which perform attestation and report the result to edge devices. Edge devices aggregate the attestation result they receive from the corresponding IoT devices, accummulate with their own attestation result, and the final attestation results are reported to the central ASSURED components. In this basic swarm attestation scheme, the attestation result enables the *centralized Verifier* to validates the *overall* state of the network. Such attestation outputs a Boolean result (e.g., 1 if all devices are healthy and 0 otherwise) without precisely identifying compromised devices. This swarm attestation scheme aims to provide privacy-preserving guarantees for both IoT and edge devices.

The Edge Devices perform one-to-one attestations of the Provers; ASSURED allows any one-to-one RA scheme to be used. A Prover is potentially untrusted and is registered with one Edge Verifier when it enters the network. Nevertheless, a Prover can be a mobile device that is temporarily unavailable to the Edge Verifier that registered its participation in the swarm. When a swarm node is mobile (to be completed in the second release of ASSURED SA - Section 6.4) and moves between the coverage of the Edge Verifiers, redundancy is introduced through the consecutive one-to-one RA of the swarm node by different Edge Verifiers. By geographically spreading the Edge Verifiers, the entire swarm network is covered. In case a swarm node is unavailable to all Edge Verifiers, the Edge Verifier that performed the last successful attestation keeps track of the timestamp of that attestation. It is up to the root verifier's policy to take action when a swarm node is unavailable for a longer time.

The Edge Verifiers keep track of the integrity of the Provers that they attest. Consequently, the root Verifier can connect to any of the Edge Verifiers to request the status of the entire network. We assume that the Edge Verifiers have sufficient storage to keep track of the attestation status of each individual swarm node, ensuring the highest level of quality of service. In ASSURED, we make no assumptions on the trustworthiness of the Edge Verifiers.

Chapter 3

Adversarial Model & Assumptions

After having defined all the possible **system and network topologies** that the final ASSURED swarm attestation scheme needs to support in the context of the envisioned use cases (Section **??**), starting with the first version that operates within a static topology (Chapter 6), and the **properties of interest** that the ASSURED swarm attestation needs to achieve, in what follows we highlight the considered adversarial model and the assumptions we make on the capabilities of each one of the devices in the swarm.

The ultimate goal of an adversary is to compromise devices and networks in order to manipulate the correct **configuration and execution of the target device, steal confidential data, get unauthorized access, etc.** In line with the adversarial model presented in D1.3 [22] and D2.1 [24], we mainly focus on **software-based attacks** and **networking attacks that try to compromise the integrity of the exchanged data** and we do not consider when a device in the swarm might get compromised; this might happen at any point in the device's lifecycle which should be detected by the ASSURED attestation enablers employed as part of the swarm attestation scheme.

Once a device is compromised, the adversary will have **full control over the software executing on the device**, such as schedule interrupts at will, accessing readable storage (e.g. ROM), modifying the writable storage (e.g. flash, RAM, and cash) and/or been able to extract cryptographic material used by any of the software functions by snooping on the memory stack. More details on such sophisticated types of attacks have already been provided in D3.4 [26] and in [33]. The only restriction, as described in Section 3.2, is that the **attacker cannot compromise the Trusted Computing Base (TCB) of each swarm device and all the software running within**; essentially, the ASSURED Tracer that is responsible for monitoring the appropriate system properties to be considered during attestation [26]. Also, we assume that the adversary will not perform hardware modification to the devices in the swarm, which means that it will not change the digital logic, exchange new CPU or add more memory. However, it is possible for an adversary to adopt some external hardware support to conduct some attacks for remote attestation. This means that the adversary considered can temper the software of any targeted device in the swarm.

Furthermore, we make no assumptions on an adversary been able to **switch the host Trusted Platform Module** (TPM as the root-of-trust of the ASSURED Wallet [29]), of the target device, with another valid TPM of her control. In this context, ASSURED is already working on the design of a protocol that will be capable to bind a specific TPM to a device CPU that will be verified during bootup: This essentially will leverage a challenge-response scheme to take place between the BIOS and the TPM registers through the UEFI Driver (Figure 3.1). more details about this scheme will provided in D3.3.

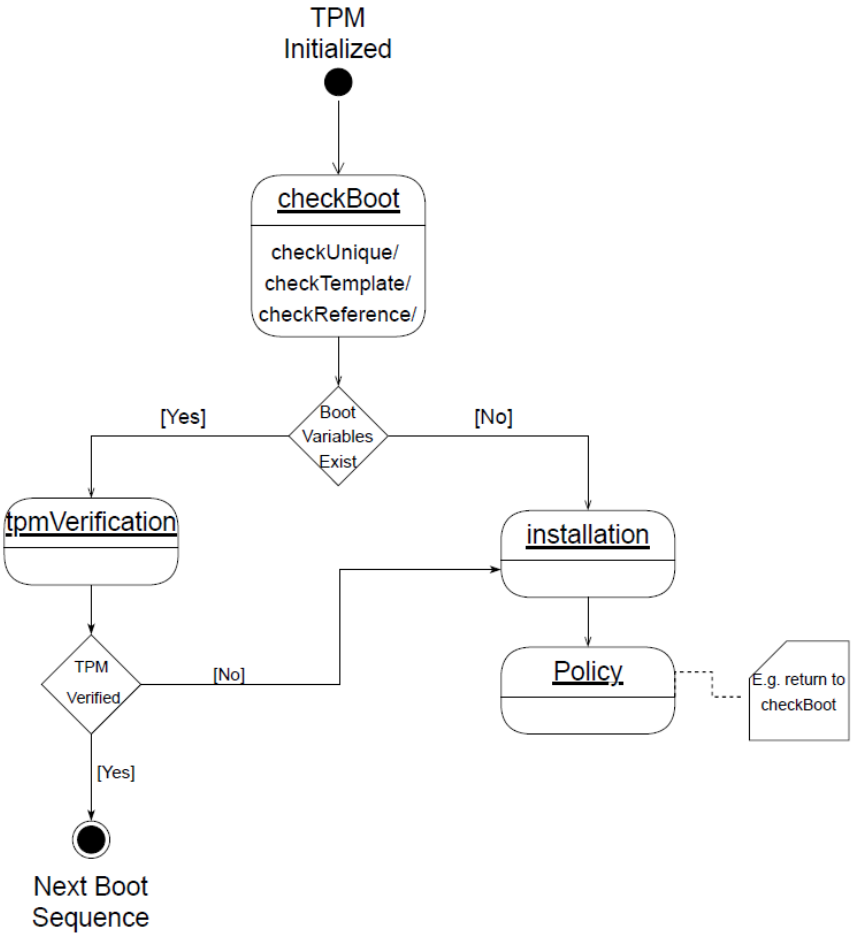

Figure 3.1: Binding a TPM to the underlying CPU

We make no assumption over the Verifier. As described in D3.2 [20], the status of the Verifier is also attested by the Blockchain Peer every time it records an attestation result on the ledger. Furthermore, in case of a dispute, this is where the employment of the **Jury-based Attestation** will be triggered so as to be able to identify the falsifying device. We assume that all the devices (Prover or Verifier) in the swarm are ordinary low-end embedded devices. If a device is compromised, the adversary can control all the activities and obtain the private message stored in the device (containing the shared secret key between neighbors), except the activities and content protected by the secure hardware. Different from device attestation, we need to conduct remote attestation over the swarm. Thus, as will be described in the next section, we also consider the **unauthorized swarm attestation attack** and **selective attestation jamming attack**.

3.1 Threat Model

In continuation to the above, we consider the usual Dolev-Yao model [34] for an adversary sitting between a resource-constrained device and edge device or between an edge device and the Security Context Broker (SCB) (Figure 3.2). This is because we assume that there is a trusted infrastructure supporting the ASSURED Blockchain components/entities, thus, the SCB is considered trusted and not a viable target for the adversary. Consequently, we consider four processes in our model [35], the Device, the TPM-based Wallet, the Edge and the SCB, and the

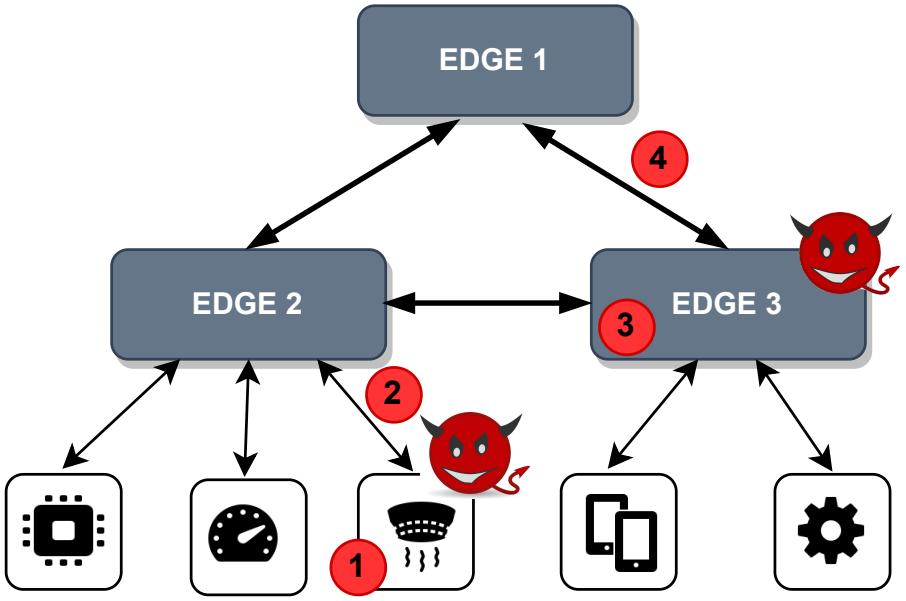

Figure 3.2: Adversary Model of Swarm attestation in ASSURED

adversary is allowed to monitor and interfere in the communication between the Devices and the Edge and between the Edge and the SCB. However, the communication between the Device and the TPM-based Wallet is treated independently. More precisely, we allow the adversary a certain degree of control over the communication channel between the Device and the internal TPM. For instance, the adversary can behave as a *passive adversary*, in order to capture more relaxed trust assumptions, such as the case where the Device has malware installed for monitoring the interactions between the (host) Device and the TPM but not for intercepting or querying the TPM (using the TPM as an *oracle*).

Assuming near zero-trust assumptions for the Device, requires additional validation of several processes that are executed in the host to protect against fully compromised Devices that try to either manipulate the parameters given to the TPM or use the TPM as a “crypto engine”; i.e., send falsified data (that can be considered as part of the attestation) to the TPM to be signed. This essentially considers a Dolev-Yao adversarial model, which allows an adversary to both monitor and modify all interactions between the Device and the TPM. The critical operation to verify is the management of the policies depicting the correct PCR values to which the use of the Attestation Key is bounded to (a compromised host can either create a false policy or bound the correct policy to a wrong representation of the PCR values, thus, resulting to the incorrect use of the AK) [18, 28]. Such verification proofs can be done using the TPM’s certification functionality and validated by a trusted entity, such as the PCA. PCR values representing the “correct” state of the Device should be circulated by the SCB, and signed under it’s private key, as part of the answer to the activate-credential command. Since the certification of the TPM has already been included in the described models, for simplicity we have excluded the further modelling of such additional proofs, thus, considering only attackers that can monitor but cannot modify the interactions between the Device and the TPM.

We have to note that this adversarial model will be adopted for all swarm attestation schemes, to be developed in ASSURED, and will be formally verified against the trust model defined in D3.1 [18]. Details of the formal verification models will be included in the next version of this deliverable (D3.7). We will consider formal verification of the attestation protocols, which allows one

to prove correctness using formal logic and mathematically rigorous arguments. Formal verification will be applied to top-level swarm attestation protocols, and we use a correct-by-construction methodology to develop executable implementations. Then simulation will be used to show applicability of the swarm attestation protocols across the envisioned applications. Currently, we have prompted to provide a qualitative security analysis (Chapter 7) of the first release of the ASSURED swarm attestation protocol against this threat model and for the security and trust requirements defined in Chapter 4.

In the following Table 3.1, we put forth a concrete list of an adversary's capabilities:

Attack	Description
Remote Software adversaries (Adv_{SW}):	The remote attacker exploits program vulnerabilities by exploiting memory-related loopholes embedded in system code, such as memory corruption vulnerabilities, broken authentication, etc. Typically, such an adversary violates the confidentiality and the integrity of the system. The most prominent types of attacks include **code injection** and **code reuse attack** [36, 45].
Code Injection Attack	This type of attack targets the exploitation of missing memory bound checks and allows the adversary to inject malicious code while providing user input. Injected code will be stored and executed within the address space of the vulnerable application. This is considered a rather impactful attack, and is the main attack vector to be detected by the CFA, as it changes the control-flow graph of the target software function.
Code Reuse Attack	During this type of attack, an adversary hijacks or overwrites code-pointer to alter the control-flow of the target device without injecting any new malicious code. The code-reuse attacks manipulate the intended behavior of the target device using various techniques such as **Return-Oriented Programming (ROP), Jump-Oriented Programming, and Counterfeit Object-Oriented Programming (COOP)**. By reusing existing code, the attacker tries to create *gadgets* which essentially comprise sets of low-level instructions (at an assembly level) that simulate the attack path that the adversary wishes to perform [37]. Furthermore, such attacks can also be used for getting access to the common memory region of a device (e.g., stack or heap structures) in order to be able to *snoop* and extract on any sensitive cryptographic material [33].
Network Adversaries (Adv_{NET})	The adversary usually performs network attacks to alter, destroy, steal sensitive data or gain access to internal systems. The most prominent type of a network attacker in the context of swarm attestation is **man-in-the-middle-attack** which, in turn, can enable the launching of unauthorized swarm attestation attacks by attackers trying to masquerade the existence of a compromised node; or by attackers that try to get access to the attestation evidence extract by a resource constrained device, thus, breaching its privacy requirements (Chapter 4).
Man-in-the-Middle attack	Such an attacker can have full control over the communication channel between two devices in the swarm e.g., the adversary can forge, drop, delay, eavesdrop the messages exchanged among two devices. As aforementioned, in ASSURED, we consider the usual Dolev-Yao model [34] for an adversary sitting between a resource-constrained device and edge device or between an edge device and the Security Context Broker (SCB) (Figure 3.2).
Physical intrusive adversaries (Adv_{PHYS})	The attacker has the ability to physically capture a device and clone it or extract secrets from it. In addition, the attacker has the ability to physically modify its components or add external hardware to the device and change its initial structure. The only restriction here is that the adversary cannot extract any information (or manipulate any of the processes) as part of the devices TCB comprising the Trusted Execution Environment and the TPM.

	Essentially, this means that the adversary cannot manipulate the Tracer running in the TCB or extract any of the keys created/loaded as part of the TPM-based Wallet (of course, she can invoke the TPM to try and use it as an *oracle*, however, this is already protected through the key usage protocols already defined during the device registration and enrollment phase). While such an attack vector also includes the possibility of **side channel attacks**, this is something that is considered outside the scope of ASSURED.
Unauthorized Swarm Attestation Attack	This attacks tries to exploit the possible lack of appropriate authentication schemes in order to have unauthenticated (deployed) devices that, at some point, may be requested to be part of a swarm attestation protocol instantiation. Especially, considering that in such decentralized schemes and dynamic topologies, we can have multi-hop devices acting as the Verifier. In ASSURED, as will be described in Chapter 7, all developed SA schemes can protect against such attacks due to the use of the TPM-based Wallet for identity management and the creation of the necessary signatures: in case of static topologies, the Edge Device authenticates each one of its children sensors prior to send the aggregate attestation result to the SCB as the Verifier; in case of dynamic topologies, each Prover device will be authenticates based on the signing of all exchanged communication by the AK (or the DAA Key in case of privacy-preservation) that was created and certified by the Privacy CA. Thus, no malicious node can then claim the coming of new swarm attestation, which can be used to conduct energy exhausting attack.
Selective Attestation Jamming Attack	Selective attestation jamming attacks are conducted by compromised devices, in the swarm, that act as the Verifier. Essentially, in an attack strategy, where the Verifier (or the attestation results aggregator) wishes to masquerade the existence of malicious Prover devices, then it can try to selectively remove their attestation result and keep in the aggregate report only the results of honest devices. However, as the ASSURED SA scheme (Chapter 6) is based on the use of (DAA-based) group signatures where the Edge Device can sign on behalf of a group based on a policy that was initiated by the SCB (which is a trusted entity), then if an attestation result is missing from the overall aggregate signature, this can be identified by any external entity verifying the signature.

Table 3.1: ASSURED Swarm Attestation - Types of Attacks Considered

Based on the above, in ASSURED, we focus on software-based attacks and consider the following attack scenarios in a swarm as depicted in Figure 3.2 :

1. A code injection adversary can modify the software configuration of any IoT device in the swarm. To avoid detection of a compromised device, the adversary will try to use an old signature σ_{old} previously generated by the IoT device on the good software configurations.

2. An adversary that compromises an IoT device will try to sign its attestation result with the key of an honest device. In particular, the adversary can impersonate an IoT device making a genuine device to send an invalid signature and use genuine devices to compute the signature for a compromised device.

3. A compromised edge device will try to produce signatures on behalf of other edge devices or their children IoT devices. Two or more compromised devices can collude together to generate signatures that cannot be traced to any of them.

4. An adversary that compromises an edge device will try to reveal the identity of the other devices in the swarm and try to distinguish the signatures created by two different honest devices.

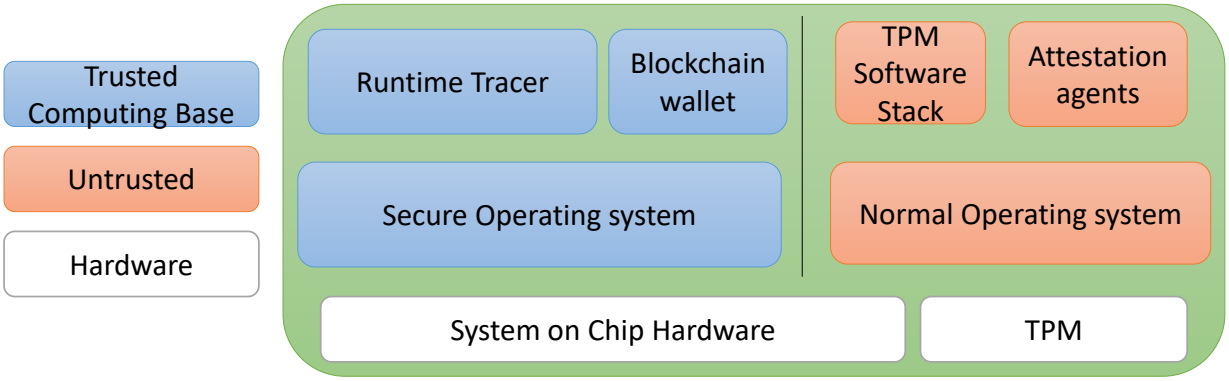

Figure 3.3: ASSURED Edge Device Architecture

3.2 Assumptions on Device Architecture

As aforementioned, the only restriction and assumptions we make is on the trustworthiness of the TCB running at the edge device. This is inline with the overall device architecture that was defined in D3.2 [20] where we described how each ASSURED edge device would contain in their package the following hardware components: **a Trusted Platform Module (TPM), a TrustZone-supported ARM processor, volatile memory, and non-volatile storage, and finally a network interface card.**

In what follows, we summarize all these hardware and software components as depicted in Figure 3.3.

3.2.1 Hardware Components

System on Chip (SoC): Edge devices in ASSURED contain at least the following hardware components: an ARM processor (supporting the TrustZone technology), random access memory, network interface, and non-volatile storage. To support efficient and fine-grained control-flow acquisition we also consider the processor to support the CoreSight technology with the program trace macrocell.

Trusted Platform Module (TPM): ASSURED relies on hardware-enforced Trusted Execution Environments (TEEs), specifically, on the TPM technology. The TPM is a hardware chip that serves as the root of trust of a platform and the core building blocks of the TPM-based Wallet that is created in ASSURED for secure key management and secure on-chain interactions with the distributed ledgers (for both reading the "*attestation smart contracts*" but also recording the results of an attestation process). The TPM creates attestations about the state of the edge devices, e.g., certifying the boot sequence. These attestations are achieved through ordinary signatures or through complex anonymous signatures known as Direct Anonymous Attestation.

ARM TrustZone: In ASSURED, we use the TrustZone architecture to provide strong security guarantees for the ASSURED tracer that enables control-flow attestation. ARM TrustZone is centered around splitting the system into a secure and a normal (non-secure) world and enforcing this separation by hardware. The hardware ensures complete isolation between these two worlds such that the programs executing in the normal world cannot leak or tamper with code or data that is part of the secure world. In TrustZone, the main memory is partitioned across the worlds, such that each world effectively has access to its dedicated memory range.

3.2.2 Software Components

Real-time Monitoring & Tracer [26]: The tracer is a software component, which is part of the Trusted Computing Base (TCB) [18] of the edge device. The tracer continuously introspects the programs executing and collects information to be used as part of the attestation schemes of ASSURED: control flow graphs for the control flow attestation and hashes of code pages for the configuration integrity verification. The tracer signs the traces in the secure world and passes them to the attestation agents to perform the requested attestation operation.

TPM-based Blockchain Wallet: The TPM will be coordinated by the host software towards forming the TPM-based Blockchain Wallet in ASSURED framework. Blockchain wallets are the central building block of ASSURED and form the basis for enhanced security, privacy and reliability guarantees for ledger management and maintenance. ASSURED blockchain wallets communicate with the rest of the Blockchain components of the ASSURED ecosystem to perform secure access control to the block chain ledger. In ASSURED, the TPM-based Blockchain Wallet is linked with the DAA towards the protection (through hw-based keys) of the verifiable credentials and attributes that are required by the devices and users for securely interating with the Blockchain infrastructure [19]. We enable the use of crypto operations of the DAA (ECC, Blind Signatures, Zero Knowledge Proofs) towards providing additional trust assurances. A DAA scheme enables a signer to prove the possession of the issued credential to a verifier by providing a signature, which allows the verifier to authenticate the signer without revealing the signer's identity.

TPM Software Stack: In the context of ASSURED, the TSS provides the necessary standard API stack for "driving" the TPM towards supporting features such as registration, login and authentication using trusted hardware/wallet for accessing the ledger, smart contract read/write/execution, management of proofs, e.g., identity, reputation value, status, in consensus algorithm. The TSS is a TCG software standard that provides a standard API for accessing the functions of the TPM. Any local or remote communication with the TPM-based Blockchain wallet is achieved using TSS. TSS provides all the necessary TPM commands for various crypto primitives. It also reduces the programming complexity of applications that desire to send individual TPM commands.

Chapter 4

Trust Assumptions and Security Requirements

In this section, we describe the security properties that the swarm attestation scheme in ASSURED should satisfy in order to be resilient to the adversarial actions presented in Chapter 3. The attestation of each *Prover* that belongs to the Swarm S should satisfy the security properties and trust requirements, as defined in D3.1 [18]. Specifically, the individual attestation of each *Prover* should guarantee: *Integrity*, *Authenticity*, *Correctness*, *Completeness*, *Atomicity*, and *Freshness* (Section 4.1).

These properties can be further enhanced with requirements that dwell on the **safety and time constraints** of real-time systems (as the ones envisioned in the context of the ASSURED use cases) and can be summarized as (Table 4.2): **Remote Verify Integrity, Efficiency, Swarm Topology In-dependency, Concurrency and Attestation In-dependency**.

SA Requirement	Description	Related Attacks
Remote Verify Integrity	This property relates to the ability of executing a **secure attestation scheme** (e.g., CFA, CIV, etc. [20]) as the enabler for the overall swarm attestation protocol capable of providing assurance claims, on the integrity of a swarm device, running tasks with real-time execution requirements that needs to overcome the inherent conflict between **real-time execution guarantees and integrity guarantees** for the measurement procedure to capture the Prover device's state. More specifically, this property translates to the need of the Verifier been able to make a decision on the correct configuration or execution state of a Prover based on correct and authentic measurements, as extracted by the Tracer, with verifiable evidence on their integrity. Consider, for instance, the case of *Smart Manufacturing* where the IoT Gateway needs to be able to attest the correct configuration of each Prover device (in the RTM system) [21]; collected traces need to be accompanied with strong guarantees on their correctness which the IoT Gateway can then use to proceed with the verification and create the aggregate attestation result for the entire swarm. In ASSURED, this is achieved through the use of the **TPM-based Wallet**, as the underlying trust anchor of each Prover device, that is used to **sign all the traces received only by the correct Tracer** (Section 6.2.2). All appropriate keys are created during the device registration and enrollment phase (Section 6.2.1.2) where the device certifies the validity of the TPM, with the Privacy CA, and then creates the DAA Key with which it will get authenticated (and its traces) to its parent Edge Device. In ASSURED, we don't make any assumption on the trustworthiness of the Verifier which can actually be any device of the target "*Systems-of-Systems*" and is also attested when recording the attestation result on the ledger (by the Blockchain Peer [31]).	Remote Software Attack; Network Attack; Man-in-the-Middle Attack; Unauthorized Swarm Attestation Attack; Selective Attestation Jamming Attack

Efficiency	This relates to the efficiency of the overall swarm attestation scheme, irrespective of the number of devices comprising the target swarm to be attested. As will be described in Chapter 6, besides the verification process (taking place at the Edge Device), the most resource-intensive operation that takes place during the execution of the protocol, is the creation of the **aggregate signature - from the Edge Device - and the subsequent signing of this result with its DAA Key.** Thus, we have to make sure that this process can be completed fast enough so that it can allow any entity to have the necessary information for making safety-critical decisions in real-time. For instance, in the "*Smart Manufacturing*" use case, the SCB needs to be able to make a decision on whether to stop the movement of a deployed robot (or not) based on the result of the swarm attestation scheme. **This requires a strict response on the correctness of the operational tasks that each Prover device is running.** In ASSURED, this is achieved through the following three features: (i) Optimized design of an Enhanced Direct Anonymous Attestation (DAA) scheme with minimizing the underline interactions needed with the TPM [20], (ii) the introduction of the **selective attestation** feature so as to not engage safety-critical and time-sensitive devices, in periodic attestation, if their **attestation history** (monitored through the ASSURED Blockchain infrastructure) shows that they have never been compromised, and (iii) the use of a new and efficient aggregate scheme (Section 6.2.2.3) leveraging the capabilities of the underlying TPM-based Wallet.	Network Attack; Man-in-the-Middle Attack; Unauthorized Swarm Attestation Attack; Selective Attestation Jamming Attack
Swarm Topology Independency	As described in Chapter 2, there can be multiple system topologies to cover various mobility and connectivity aspects as well as different number of Verifiers (Table 2.1) that can be encountered in complex SoS environments: either **static or dynamic topologies**. These have different requirements as it pertains to how the swarm attestation scheme will operate - especially considering that one mobile node might need to change its parent (verifying) node as it changes its position which translates to having multi-hop verification, thus, the need to protect against unauthorized attacks (Table 3.1). Therefore, it is vital that the ASSURED SA leverages advance authentication mechanisms. More specifically, through the employment of the newly designed DAA scheme, ASSURED can achieve the authenticity and establishment of trustworthy channels between all swarm devices while also preserving their privacy. This has in turn been enhanced even further so as to be able to add *linakbility* features in the DAA in order to be able to achieve **traceable privacy-preserving aggregation of all attestation results irrespective of the underlying network topology** (Section 6.2.3.2).	Network Attack; Man-in-the-Middle Attack
Attestation Independency	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 should be 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. The Verifier should be able to accommodate all these operational needs. Furthermore, the collection and tracing of the required attestation evidence should not impede with the normal operation of the deployed devices: Allowing the measurement procedure [26] to respect the real-time demands of the system's tasks could be easily misused by an adversary, e.g., to restore a benign state while the measurement is performed. Therefore, the measurement procedure must be able to capture the system state independent of the execution of real-time tasks [44]. As aforementioned, in ASSURED, the first release of the swarm attestation scheme (Chapter 6) considers the execution of the same type of attestation enabler for all devices in the swarm.	Remote Software Attack; Code Injection Attackl Code Reuse Attack

	However, this will be extended in the second release to also include attestation in-dependency and attestation evidence privacy by leveraging ring signatures (on top of the currently employed group signatures). For the latter, ASSURED non-intrusive tracing capabilities [26] ensures that all system state measurements can be captured correctly.	

Table 4.1: Swarm Attestation Generic Requirements

Additional properties that are introduced by the characteristics of the types of environments considered in ASSURED and the features of the heterogeneous devices include **Malicious Device Identification, Mobility Tolerance, Verifiable Attestation Launching, Opaque & Adjustable Privacy Level, and Attestation Evidence Granularity Depth**.

ASSURED SA Requirement	Description
Malicious Device Identification	Irrespective of the level of detail that is included in the aggregate attestation result (cf. Attestation Evidence Granularity Depth), the root Verifier - or any other device acting as the Verifier (Figure 3.2) - should be able to correctly trace back to the identity of any IoT Device with a failed attestation result which represents an indication of risk. This translates to the need of **privacy-preserving traceability**: In case of a "*true*" swarm attestation result where the root Verifier simply checks the sanity of the overall swarm network, the identities of the Edge and IoT Devices can be protected. However, in the case of a failed attestation result been detected, for one of the swarm IoT devices, it should be feasible for any external Verifier to trace back this result to the Edge Device and request to get access to the attestation result of the specific IoT Device with the failed attestation.
Mobility Tolerance	The attestation protocol should be efficient and it can tolerate device mobility to some extent. It means that the attestation scheme can efficiently and securely attest the swarm as long as the mobile node could re-join the attestation with different joining point (i.e., Edge Device). This should also be able to protect against **unauthorized swarm attestation attacks**, especially for mobile nodes that re-join to a different part of the network: **they should be verified by the new Edge Device when it comes to the validity of their identity and AK credentials**.
Verifiable Attestation Launching	Only honest Verifiers can launch a new swarm attestation. This pertains to both cases of having one root Verifier (Figure 3.2) or having multiple Verifiers in dynamic network topologies. In both scenarios, in the context of ASSURED, the swarm attestation is initiated when a respective **attestation policy is deployed on the ledger via the use of smart contracts** [19]. Thus, any external Verifier can audit whether the attestation process was initiated and executed correctly. This is also ratified through the use of an *attestation challenge* (i.e., nonce), read by the Blockchain, that is signed by both the SCb and the Edge Device when initiating the swarm attestation process.
Opaque & Adjustable Privacy Level	This property focuses on two privacy aspects that are core to our scenarios: both when it comes to **device identity privacy** and **attestation evidence privacy**. For the former, as the name suggests, it should be possible to "*hide*" the identity of an IoT Device, as part of the swarm, or protect the *unlinkability and anonymity* of the Edge Device (Section 4.1) that acts as the swarm head for aggregating the received measurements/results from the IoT Devices. For the latter, this pertains to safeguarding the information that can be implicitly extracted through the monitored system traces; either through the control-flow graphs (in the case of CFA) or device configuration (in the case of CIV). Especially in the case of CFA, giving full knowledge (to the Verifier) of the full program execution details and in-memory layout (through the control-flow graphs that are included in the attestation evidence) constitutes a privacy breach:

	implementation disclosure attacks are feasible where "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. Thus, ASSURED should guarantee that the root Verifier should not be able to have access to the internal details of the attestation evidence but only on the attestation result: **have verifiable evidence on the correct execution or configuration of each swarm device but without knowing what is the exact configuration or execution profile**. *This is a rather challenging task and it will include the employment of advanced ring signatures (scheduled for the second release)* (Section 6.3). More details on the supported levels of privacy considered in ASSURED can be found in Section 4.2
Attestation Evidence Granularity Depth	This pertains to the level to which the attestation information is refined by the root Verifier (Figure 3.2). Recall that in ASSURED we are dealing with devices, as part of a service graph chain, that need to establish trust relationships for cooperatively executing (safety-critical) functions. In this case, the SCB (or any other authorized root Verifier) should be able to attest the entire network: as a **binary result**. Either all the device of the service graph are ''*as expected*" or not. Consider again, for instance, the case of "*Smart Manufacturing*" where the IoT Gateway is interested in the correctness of the entire swarm of devices comprising the RTM. **This usually considers the attestation of the same type of properties for the entirety of the swarm.** However, in case that different properties need to be attested (which will result in different representations of the Platform Configuration Registers (PCRs of the TPM-based Wallet)) or if the operational needs of the application dictate for the root Verifier to be able to have access to each devices' attestation result, then this should be possible to trace it back, first, to the Edge Device and then the IoT Device. Thus, starting from "*top-to-bottom*": high-level granularity should reflect the state of the overall swarm whereas low-level granularity should enable the possibility of the root Verifier requesting access to the attestation result of a specific device. For the former, the root Verifier should receive the aggregate result (Section 6.2.2.3), of each Edge Device, that represents the attested state of each IoT Device - with strong guarantees on the integrity of the aggregate result. For the latter, in the case of a failed attestation result, the root Verifier should be able to trace to the Edge Device (as the Data Aggregator) and request for the attestation result of any of the IoT devices (Section 6.2.3.2). This essentially enables the **traceability** feature: *attestation results should adhere to the required privacy level (cf. **Opaque & Adjustable privacy Level**) but should also allow for tracing back to the origin of the attestation when needed (failed attestation).*

Table 4.2: ASSURED-Specific SA Requirements

4.1 Security Properties

In addition to the aforementioned functional specifications and trust requirements, the swarm attestation in ASSURED should guarantee the following fundamental security properties:

1. ***Existential Unforgeability against Adaptive Chosen Message and Chosen Public-Key Attacks***: It should be computationally infeasible to produce message signature pairs (M;σ) that are accepted by verification algorithms without knowledge of the device's secret key. This essentially translates to only authenticated devices been able to participate in the swarm attestation protocol and that no impersonation and unauthorized SA attack can take place. In ASSURED, this is supported by the use of the TPM-based Wallet that protects all the cryptographic material to be used throughout the SA protocol.

2. ***Linkability***: Two signatures are linked if they are generated using the same private key

or the same $basename$ in case DAA SIGN is been used (Section 6.2.2.3). The former represents the possiblity to link signatures back to the IoT Devices - as will be described in Section 6.2.2.1, IoT Devices use short-term anonymous credentials to sign their traces to be sent to the Edge Device. The latter pertains to the Edge Device that acts as the data aggregator and signs the aggregate result using their DAA Key.

3. ***Anonymity***: An adversary that is given two signatures with different tags (signed under the same DAA Key using a different $basename$), cannot distinguish whether both signatures were created by one honest platform, or whether two different honest platforms created the signatures.

4. ***Provenance***: A Verifier (Vrf) of a swarm can verify that the aggregate result is based on the target devices, comprising the swarm of interest, by certifying either the identity of each device (no privacy protection is needed) or based on the correct number of the group members (check the number of signatures included in the aggregate result) without though been able to link parts of the signature to the original device creators. Linkability should be enabled only in the case of a failed attestation result.

5. ***Non-frameability***: No adversary can create signatures on a message that links to an honest device when this honest device never signed it.

6. ***Traceability***: The group-based signature should be traceable to the group member who issued it. This should be feasible only if verifiable evidence can be produced on the existence of a failed attestation result, thus, with the support of the SCB any external Verifier can tace back the signature to the Edge and/or IoT Device (Section 6.2.3.2).

7. ***Coalition resistance***: The possibility of a group of signers colluding together to generate signatures that cannot be traced to any of them.

8. ***Exculpability***: This is the property that no member of the group and not even the group manager can produce signatures on behalf of other group members.

9. ***User Revocation*** A group manager should be able to correctly revoke a corrupt signer as this is might relate to a device that has been compromised.

10. ***Integrity of Swarm Attestation as a Whole:*** The aggregated attestation result should reflect the claimed integrity measurement during the attestation.

11. ***Privacy Preservation:*** Each interacting Prover should authenticate itself in a privacy-protecting way. The Verifier should be able to verify the attestation result without having detailed knowledge of the Swarm.

12. ***Integrity of exchanged communication data among devices.***

4.2 Privacy Requirements

As aforementioned, there are two privacy considerations that need to be tackled, in complex systems, when trying to concurrently attest a swarm of devices: **Device Privacy Identity** and **Attestation Evidence Privacy**.

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 [22]) 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**.

To enable a Prover to convince a Verifier about the possession of secret information without revealing any additional knowledge, *zero-knowledge Succinct Non-interactive Arguments of Knowledge* (zkSNARK) [38, 43] has seen a grown research interest in both academia and industry. However, despite various improvements in their implementation [2, 3, 47, 48], the zkSNARK design remains prohibitively expensive, especially for resource-constrained Provers. Similarly, while TEEs, such as Intel's SGX, can also achieve verifiable computation (VC) in zero-knowledge similar to zkSNARKs as demonstrated by [46], porting arbitrary software into an SGX enclave is difficult, and TEEs generally incur significant performance overhead. Therefore, in the context of designing a privacy-preserving CFA protocol, there are still a number of challenges to overcome.

in the context of ASSURED, this is one of the core features to be enabled within the second release of the CFA and swamr attestation schemes to be designed by M30. As described in Section 6.4, such privacy-preserving attestation capabilities will be provided through the use of zkSNARK circuits embedding the claim that a program's execution path is benign as a bundle of mathematical statements in an arithmetic circuit, which the Prover can prove in zero-knowledge.

Regarding **Device Privacy Identity**, this is a double-edge sword: On one hand, we need to **protect the system from malicious devices** by identifying the origin of a possible compromise, thus, revealing the identify of a device with failed attestation result. On the other hand, however, we should be able to also **protect devices from untrusted verifying entities**, especially those where the attestation history shows that they have never been compromised or rarely so, by enabling *unlinakbility*: **No successful attestation result should be linked back to its device origin** as this could enable attacks like the ones mentioned before; i.e., implementation disclosure attacks. Or even enable "*honest-but-curious*" entities that do not want to impede with the protocol functionality but are interested in creating device attestation profiles - *how often a device is been required to execute an attestation task (might reveal type of safety-critical functions executed), the type of attestation task, the set of devices comprising the entire swarm, etc.*

In the context of ASSURED, in order to be able to achieve this privacy requirement, we have designed a **Traceable DAA scheme** (Chapter 6) where the anonymity, unlinkability and untraceability features of DAA hold until an authenticated (verifying) entity can present the nec-

essary evidence for getting access to the identify of a device with failed attestation result. For this to materialize, the SCB serves as the *Tracing Entity* that can provide this linkability (Section 6.2.3.3). Furthermore, based on the operational needs of a scenario, we have defined the following privacy-preserving profiles to be achieved during the execution of the ASSURED SA protocol (Section 6.2.2.2):

- **Full Anonymity (Privacy-Preserving SA):** As described in Section 2.2.1, the ASSURED SA topology considers Edge Devices that serve as data aggregators receiving the authentic attestation results from IoT Devices and merge them together to an aggregate result to be shared with the root Verifier. In this context, *full anonymity* relies on the fact that Edge Devices should not merge the signed (failed attestation) result of an IoT Device in the aggregate signature. Thus, the root Verifier (and any other external verifying entity) will understand that one (or more) IoT Devices failed with the execution of their attestation process from the size of the aggregate signature (number of included signatures would be less than the original size of the swarm). In this case, **only those verifiers with the appropriate authentication privileges can interact with the SCB to link back a failed attestation result to the identify of the origin IoT Device.**
- **Partial Anonymity (Privacy-Preserving SA based on *Pseudonymity*):** As will be described in Chapter 6, ASSURED SA employs group-based signatures supported by the Enhanced DAA scheme [20] through the use of short-term anonymous credentials; i.e., *pseudonyms* used by the IoT Devices to anonymously sign their attestation results. In this context, and for such a privacy protection profile, the failed attestation results (of any IoT Device) will be merged to the overall aggregate signature created by the parent Edge Device. However, the identity of the IoT Device will be protected since it will not be recorded on the ASSURED DLT. In contrast, the aggregate result will contain the attestation result signed by the IoT Device's pseudonym which can only be linked by the partner Edge Device. Therefore, if an **authenticated Verifier wishes to link back to the device identity, it first needs to interact with the SCB (as the *Tracing Entity*) to de-anonymize the Edge Device that performed the attestation data aggregation which, in turn, holds the mapping of each pseudonym belonging to a specific IoT Device.**
- **No Anonymity:** In this case, there is no device identity privacy that needs to be preserved. Hence, the Edge Device upon reception of a failed attestation result (from an IoT Device), it will both merge this result in the overall aggregate signature and will **record the identity of the IoT Device on the ASSURED DLT.**

Chapter 5

ASSURED Building Blocks

As mentioned in Chapter 2, in the context of the newly designed ASSURED swarm attestation scheme, a spanning tree structure is adopted where parent (edge) devices can collect the attestation results of their child devices and then relay the overall attestation report to the SCB. Based also on the previously described *security, privacy, and trust* requirements, it is important to be able to **hide the identities of the devices comprising a swarm and identity resolution should only be possible in the case of a failed attestation result**, thus, enabling **malicious device identification** which is important for improving the survivability of a swarm through revoking the credentials of compromised devices or re-deploying new attestation policies. This is achieved through the use of **group-based signature schemes enhanced with a variant of the DAA scheme capable of also providing privacy-preserving traceability**. Also, we need a non-repudiation way to guarantee that a malicious node can not forge a fault attestation result of a device and the attestation result collected by a device (even malicious) can not be denied. This achieved through the signing of all system measurements, from the IoT devices, by leveraging **short-term anonymous credentials (i.e., pseudonyms)** already registered to their parent node. Finally, to prevent a malicious device from launching fake attestation to its children devices, we need an **efficient and verifiable attestation launch guarantee** which is achieved by the initiation of the process through the circulation of an attestation challenge that includes the nonce extracted from the ledger (as part of the $createNonce$ function of an attestation contract [31]) and is signed using its DAA Key binded to the underlying TPM-based Wallet.

In what follows, we give all the foundation behind the signature-related cypto primitives used for the design of the ASSURED SA protocol.

5.1 Aggregate Signatures

As presented in [5–7], the ultimate purpose of designing aggregate signatures is to reduce the length of digital signatures in applications that use multiple signatures. An aggregated signature, given n signatures on n distinct messages generated by n distinct users, combines this collection of signatures into a single short signature σ. The length of this aggregated signature σ should be the same as a signature on a single message. Overall there are two mechanisms for signature aggregation: (1) **general aggregation** (e.g., [6]) and (2) **sequential aggregation** (e.g., [32, 42]). In a **general aggregate signatures** scheme, each user i signs her message M_i to obtain a signature σ_i. Then the aggregation can be done by anyone and requires no secret keys. So anyone can use a public aggregation algorithm to take all n signatures $\sigma_1, ..., \sigma_n$ and compress them into a single signature σ. To verify an aggregate signature, all the original messages and public keys

are needed. Specifically, an aggregate verification algorithm takes $PK_1, ..., PK_n$, $M_1, ..., M_n$, and σ and decides whether the aggregate signature is valid. We refer to this mechanism as general aggregation since aggregation can be done by anyone and without the cooperation of the signers.

In a **sequential aggregation** scheme, signature aggregation can only be done during the signing process. Each signer in turn sequentially adds her signature to the current aggregate. Thus, there is an explicit order imposed on the aggregate signature and the signers must communicate with each other during the aggregation process. For instance, user 1 signs M_1 to obtain σ_1; user 2 then combines σ_1 and M_2 to obtain σ_2; and so on. The final signature σ_n binds user i to M_i for all $i = 1, ..., n$.

Multi Signatures: In addition to the aforementioned mechanisms, a related concept to aggregate signatures are also multisignatures. While aggregate signatures let anyone aggregate signatures from different signers, multisignatures lets anyone aggregate signatures from a single signer. In multisignatures, a set of users all sign the same message and the result is a single signature. A multisignature scheme allows any subgroup of a group of players to jointly sign a document such that a verifier is convinced that each member of the subgroup participated in signing.

In the following, we summarize the detailed algorithms of both aggregate signatures mechanisms, namely general and sequential aggregation, based on the works presented in [5–7, 32, 42].

5.1.1 General Aggregate Signatures

In a general aggregate signature scheme, signatures are individually generated and then combined into an aggregate. The aggregation algorithm takes as input signatures $\sigma_1, ..., \sigma_n$ on respective messages $M_1, ..., M_n$ under respective public keys $PK_1, ..., PK_n$. (The assignment of indices is arbitrary.) It outputs a single aggregate signature σ whose length is the same as a signature on a single message.

The aggregate verification algorithm, given an aggregate signature σ, public keys $PK_1, ..., PK_n$ and messages $M_1, ..., M_n$, verifies whether the aggregate signature σ is valid.

5.1.1.1 Bilinear Aggregate Signatures

The bilinear aggregate signature scheme , [6] enables general aggregation. An arbitrary aggregating party unrelated to, and untrusted by, the original signers can combine pre-existing signatures into an aggregate. The system does not impose an order on the aggregated elements. The scheme includes the three usual algorithms for generating and verifying individual signatures, as well as two additional algorithms that provide the aggregation capability.

Key Generation. For a particular user, pick random $x \leftarrow \mathbb{Z}_p$ and compute $v \leftarrow g^x$. The user's public key is $v \in G$. The user's private key is $x \in \mathbb{Z}_p$.

Signing. For a particular user, given the public key v, the private key x, and a message $M \in \{0,1\}^*$, compute $h \leftarrow H(v, M)$, where $h \in G$, and $\sigma \leftarrow h^x$. The signature is $\sigma \in G$.

Verification. Given a user's public key v, a message M, and a signature σ, compute $h \leftarrow H(v, M)$; The signature is accepted as valid if $e(\sigma, g) = e(h, v)$ holds.

Aggregation. Arbitrarily assign to each user whose signature will be aggregated an index i, ranging from 1 to n. Each user i provides a signature $\sigma_i \in G$ on a message $M_i \in \{0,1\}^*$ of her choice. Compute $\sigma \leftarrow \coprod_{i=1}^{n} \sigma_i$. The aggregate signature is $\sigma \in G$.

Aggregate Verification. We are given an aggregate signature $\sigma \in G$ for a set of users, indexed as before, and are given the original messages $M_i \in \{0,1\}^*$ and public keys $v_i \in G$. To verify the aggregate signature σ, compute $h_i \leftarrow H(v_i, M_i)$ for $1 \leq i \leq n$, and accept if $e(\sigma, g) = \coprod_{i=1}^{n} e(h_i, v_i)$ holds.

5.1.2 Sequential Aggregate Signature

In a sequential aggregate signature scheme, aggregation and signing are performed in a single combined operation. The signers sign messages one after the other. Signer number i takes as input the aggregate signature from signer number $i-1$ and adds a signature on message M_i to the aggregate. The resulting aggregate signature is given to signer $i+1$ and so on until all messages are signed. The final aggregate signature should have the same length as a signature on a single message. To verify an aggregate signature, the algorithm is same as in the general aggregate signatures. In particular, only the final aggregate is given to the verifier along with all messages and all public keys.

5.1.2.1 The Trapdoor Sequential Aggregate Signature Scheme

Sequential aggregate signatures are built from trapdoor homomorphic permutations. A permutation family Π is a collection of permutations of some domain D. Each permutation in Π has a description $s \in S$. Anyone given a description s can evaluate the corresponding permutation. A permutation family is trapdoor if each description s has some corresponding trapdoor $t \in T$ such that it is easy to invert the permutation corresponding to s with t, but infeasible without t. (S and T are arbitrary sets) The trapdoor sequential aggregate signature scheme is related to the fulldomain hash signature scheme and is based on a multisignature scheme.

This scheme employs a fulldomain random-oracle hash function H mapping inputs into D, $H : \{0,1\}^* \rightarrow D$. A signer provides to H every public key and every message in the aggregate signature she is creating. Thus H is of the form $H : \cup_{j=1}^{\infty}[(S)^j \times (\{0,1\}^*)^j] \rightarrow D$

Key Generation. For a particular user, pick random $(s,t) \leftarrow Generate$ and compute $v \leftarrow g^x$. The user's public key PK is s. The user's private key SK is (s,t).

Aggregate Signing. The input is a private key (s,t), a message $M \in \{0,1\}^*$ to be signed, and a sequential aggregate signature $\sigma^{'}$ on a vector of messages $\mathbf{M}$ under a vector of public keys $\mathbf{s}$. No key may appear twice in $\mathbf{s}$. Furthermore, the vectors $\mathbf{M}$ and $\mathbf{s}$ must have the same length. Let i equal to the length of vector $\|\mathbf{M}\|$. If i = 0, then $\sigma^{'}$ = 1.

Compute $h \leftarrow H(\mathbf{s}||s, \mathbf{M}||M)$ and $\sigma \leftarrow Invert(s, t, h * \sigma^{'})$. The sequential aggregate signature is $\sigma \in D$.

Using π-notation, a sequential aggregate signature is of the form:

$\pi_i^{-1}(h_i * \pi_{i-1}^{-1}(h_{i-1} * \pi_{i-2}^{-1}(...\pi_2^{-1}(h_2 * \pi_1^{-1}(h_1))...)))$, where $h_j = H(\mathbf{s}|_1^j, \mathbf{M}|_1^j,)$. Here $\mathbf{s}|_1^j$ is the sub-vector containing elements $\mathbf{s}_1, \mathbf{s}_2, ..., \mathbf{s}_j$, where $1 \leq j \leq \|\mathbf{s}\|$.

5.1.3 Aggregate Signatures in ASSURED

Both aggregate signature mechanisms, namely **general aggregation and sequential aggregation**, provide the ability to aggregate a collection of signatures on a large number of messages by different users into a single short signature. In the first technique, based on bilinear maps, the

aggregation happens all at once after all the individual signatures are generated. This aggregation can be done by anyone and requires no secret keys. In the second aggregation technique, based on homomorphic trapdoor, the aggregation is done during the signing process.

In ASSURED, we follow the first technique (i.e, ***general aggregation signature scheme****) since it is easier to be adopted in the ASSURED swarm and provides more efficiency to the Edge Device that calculates the aggregate signature based on the collection of signatures from all its children IoT devices. It is as powerful of a technique and also enables the easy conversion to a sequential aggregation when needed*. The latter is an interesting feature to be explored in cases where devices, as part of an overall service graph chain, needs to communicate and exchange data (in a specific order) so as to cooperatively execute safety-critical functions. In such a setting, it is preferable to have the result of each device - as part of the overall aggregate signature - in a sequential manner so that the Verifier can check that the order of execution was correct prior to checking the correct state of each device as part of the chain; leading to the creation of a **chain-of-trustworthiness** computed and verified through the swarm attestation scheme.

Consider, for instance, the case of *Smart Aerospace* in ASSURED and specifically the scenario of **software remote update** [30] where all (or a subset) of the Electronic Control Units (ECUs) of the airplane need to securely receive a software update to load it and execute it as part of their maintenance. In this context, the ECUs comprise the *resource-constrained devices*, in the swarm topology (Section 2.2.1), while the Secure Server Router (SSR) is the Edge Device (Data Aggregator) and the Verifier is the Ground Station. Based on the functional specification and the operational needs of this scenario, there is a specific order of how the communication needs to take place between all involved parties: First, the SSR needs to be securely enrolled and attested, in the overall system, in order to make sure that it will have all the appropriate cryptographic material for been able to sign on behalf of the airplane ECUs; Second, all of the ECUs that need to perform the update need to be verified against their configuration state - that they are loaded with a specific OS version prior to be allowed to run the target update; Finally, the ECUs get the update (through the ASSURED Blockchain) and need to execute it in a specific order; e.g., the ECU operating the flaps needs to be updated prior to the ECU that controls the airplane wings. Thus, while been requested - as a swarm - to again attest to their correct state, after the update, each devices' result still needs to be recorded in a sequential manner.

Compounding this issue, the SSR should be able to use the sequential aggregation capabilities so that it can create an aggregate signature including the ECU signatures in the order that they executed the update and sent back their result. This will be considered as an enhancement in the second release of the ASSURED swarm attestation scheme.

5.2 Group-based Signatures

In the context of an Edge Device, as the data aggregator, been able to **securely receive the system measurements** (for attestation) from all its children (resource-constrained) IoT devices, but in a **privacy-preserving manner** (no measurement should be linked back to the IoT device unless requested in the context of a failed attestation result), ASSURED will leverage the use of group-signatures, as provided by the DAA protocol, enhanced with a *linkability* token for achieving **privacy-preserving but traceable aggregation**.

In what follows, we describe these two types of privacy-preserving signatures that will be used as building blocks for facilitating the ASSURED swarm attestation.

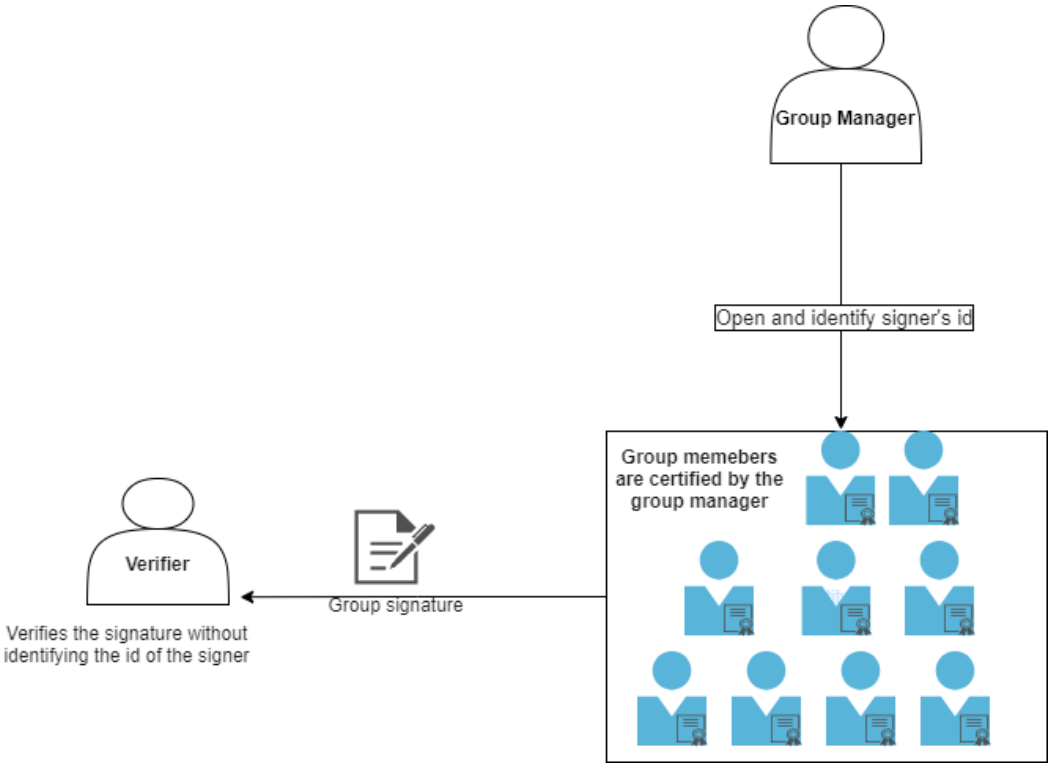

Figure 5.1: Overview of the Group Signature

5.2.1 Group Signatures

Group signature aims to provide a way to guarantee that a message was sent by a group member (authentication and data integrity) without leaking any information about which group member signed the message (privacy) unless an opening authority decides to open the signature as described in [16], [1] and [14]. More precisely, group signatures allow members of a group, which is administered by a group manager, to anonymously sign messages on behalf of that group. The signature Verifier will be convinced that the signature comes from some group member, but without knowing the signer. At the same time, an authority is able to determine the signer's identity (traceability) using some trapdoor information known as the signature opening operation.

Security properties of Group Signatures: The basic requirements of a group signature scheme is that any honest signature generated by a group member should be accepted as correct, also the signature should be traceable to the group member who issued it. Group signatures are also required to be anonymous, meaning that without the tracing key, it should be infeasible for an adversary (even given all the signing keys) to determine the identity of the group member who issued a specific signature. A second important requirement for group signature scheme is that it should be difficult for an adversary who can corrupt some set of group members C to output a valid signature that can't be traced to some member in C, this means traceability.

Group signatures fall into two categories, in terms of **group dynamicity** (which is also one of the core factors that affect the level of untraceability offered to each group member): **static and dynamic**. The former requires a fixed number of group members, whereas the latter allows dynamic addition of members to the group. The selection of the appropriate scheme is coupled to the context of the operational needs of a deployed service. To exemplify this, consider the case of *Smart Manufacturing* where a new robotic device can be added in the manufacturing floor at any point in time. This newly on-boarded device will be first securely enrolled with the

IoT Gateway and then can be added to any specific swarm of devices; e.g., to the swamt of devices that are part of a specific Real-time Monitoring System (RTM). On the other hand, in the context of *Smart Aerospace* the set of Electronic Control Units (ECUs) that are considered as part of a swamr of devices in an airplane will never change (unless manually added by the System Administrator). In this case, the number of eligible devices is known and, thus, static group signature schemes are applicable. Otherwise, dynamic group signatures are necessary. ASSURED supports both types of group signature schemes; Short Group Signatures [4] (static) and the Camenisch-Groth scheme [13] (dynamic).

Syntax of Group Signature Schemes

A group signature scheme *GS=(G.Key Gen, G.Sign, G.Vrfy, G.Open)* consists of four polynomial-time algorithms:

1. The randomized group key generation algorithm *G.Key Gen*$(1^n, 1^N)$ takes inputs $1^n, 1^N$, where $n \in \mathbb{N}$ is the security parameter and $N \in \mathbb{N}$ is the group size. This algorithm returns (PK, TK, gsk), where PK is the group public key, TK is the group manager's tracing key, and gsk is a vector of N signing keys such that $gsk[i]$ corresponds to the i^{th} user signing key.

2. The group signature algorithm *G.sign*$(gsk[i], m)$ is a randomized algorithm that takes the i^{th} user secret key $gsk[i]$ and a message M, outputs a signature σ on m.

3. The group signature verification algorithm *G.vrfy*(PK, m, σ) is a deterministic algorithm that takes the group public key PK, a message m and a signature σ on m, returns 0 or 1.

4. The opening algorithm *G.Open*(m, TK, σ) is a deterministic algorithm that takes a message m, the tracing key TK and a signature σ as inputs, and returns an identity $i \in [N]$.

The general security notions for group signatures based on [8] are:

- **Correctness**: Correctness guarantees that an honest user can enroll in the group and produce signatures that are accepted by the Verify algorithm.
- **Anonymity**: Group signatures should be anonymous and don't reveal the identity of the group member who produced them. Since the group manager has the ability to trace signers, we must assume the group manager to be honest for anonymity to hold, but some of the other users may be malicious.
- **Traceability**: Traceability in group signatures was purely with respect to identifying a signer but did not require a proof of correct opening. Bellare et al. [1] included the use of a proof for correct opening that can be verified by anybody using a Judge algorithm. Eliminating the proof of correct opening can be done in case of putting trust on the group manager.
- **Opening Soundness**: This property guarantees that the opening of any group member signature's should not be attributed to somebody else.
- **Opening Binding**: This property guarantees that even if all authorities and users collude they should not be able to produce a valid signature that can be selectively traced back to different members.

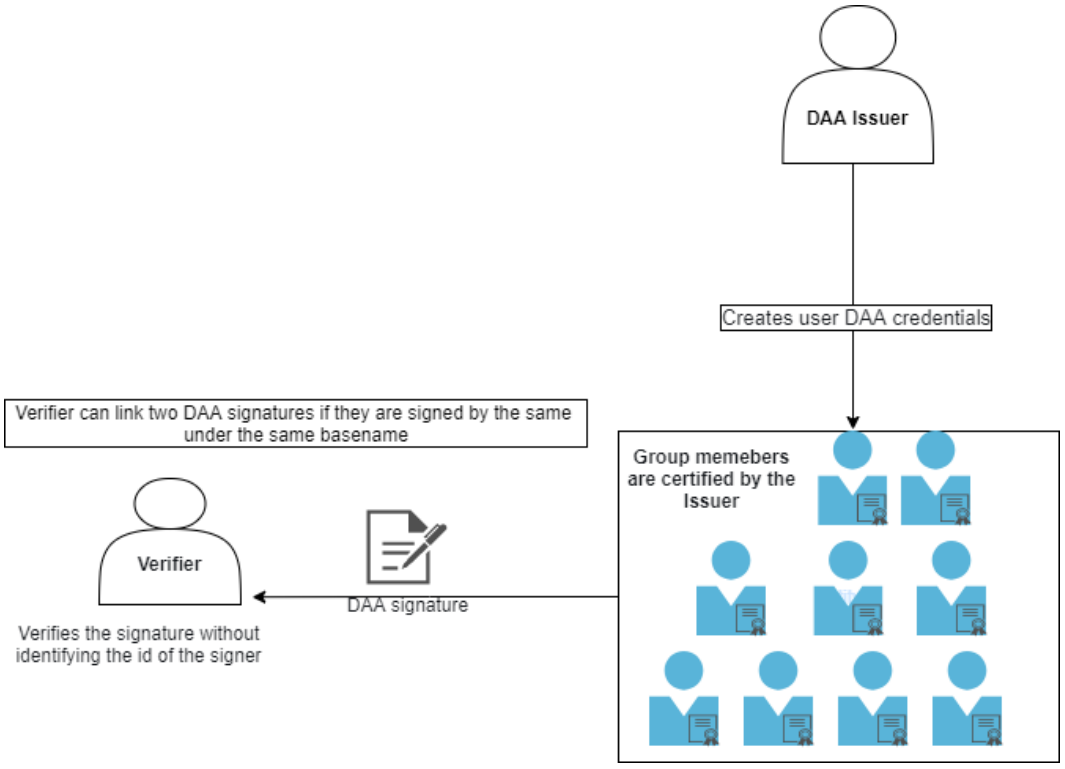

Figure 5.2: Overview of the DAA Signature

5.2.2 Direct Anonymous Attestation (DAA)

In general, a Direct Anonymous Attestation (DAA) scheme consists of an issuer(Privacy CA), a set of signers and a set of Verifiers. The issuer creates a DAA membership credential for each signer.A DAA credential corresponds to a signature of the signer's identity produced by the DAA issuer. A DAA signer consists of the (Host, TPM) pair. Their membership to the DAA community and trustworthy state is proved by providing the Verifier with a DAA signature of the TPM representation of the Host state. The DAA signature includes a zero-knowledge proof-of-knowledge, which is a cryptographic construct used to convince the Verifier that the signer possesses a valid membership credential, but without the Verifier learning anything else about the identity of the signer. In contrast to other privacy-preserving constructs, like group signatures [40, 41], DAA does not support the property of traceability, wherein a group manager can identify the signer from a given group signature. In particular, the DAA is a protocol enabling a TPM and a Host device not only to authenticate themselves to a verifying edge device and to prove that the Host is in a trustworthy state, but also to do so in a privacy-preserving manner. More concretely, the DAA provides the TPM with the ability to sign its register values in an anonymous way, whilst still convincing the Verifier that it possesses valid DAA credentials.

Furthermore, when the DAA issuer also plays the role of a Verifier, the issuer does not obtain more information from a given signature than any arbitrary Verifier. However, to prevent a malicious signer from abusing anonymity, DAA provides two alternative properties as the replacement of traceability. One is the rogue signer detection, i.e. with a signer's private key, anyone can check whether a given DAA signature was created under this key or not. The other is the user-controlled linkability: two DAA signatures created by the same signer may or may not be linked from a Verifier's point of view. The linkability of DAA signatures is controlled by an input parameter called the basename. If a signer uses the same basename in two signatures, they are linked; otherwise, they are not.

5.2.2.1 Security Properties of DAA

We summarize the following high-level security properties of DAA:

Unforgeability: When the Privacy CA and all TPMs are honest, no adversary can create a signature on a message μ with respect to basename bsn when no TPM signed μ with respect to bsn.

One-More-Unforgeability: When the Privacy CA is honest, an adversary can only sign in the name of corrupt TPMs. More precisely, if n TPMs are corrupt, the adversary can create at most n unlinkable signatures for the same basename.

Anonymity: An adversary that is given two signatures, with respect to two different basenames, cannot distinguish whether both signatures were created by one honest device, or whether two different honest devices created the signatures.

Non-frameability: No adversary can create signatures on a message μ with respect to basename bsn that links to a signature created by an honest device for the same basename, when this honest device never signed μ with respect to bsn.

The anonymity and non-frameability properties must hold even when the Privacy CA is corrupt. The existing DAA schemes implemented in the TPMs are based on either the factorization problem in the RSA setting or the discrete logarithm problem in the Elliptic-Curve (EC) setting. The concept of a DAA scheme was first proposed in 2004 by Brickell, Camenisch, and Chen [9]. This scheme is called RSA-DAA and supported by the TPM version 1.2. Later, Brickell, Chen, and Li proposed the first EC-DAA scheme based on symmetric pairings [10,11]. Two EC-DAA schemes, based on asymmetric (Type 3) pairings, are supported by the TPM version 2.0 [12,17].

Chapter 6

Scalable Heterogeneous Layered Attestation

In what follows, we provide a high-level overview of the ASSURED *swarm attestation* scheme focusing first in static network topologies[1]. Based on the building blocks described in Chapter 5, the first version will leverage the capabilities of **group signature schemes as offered by the ASSURED Direct Anonymous Attestation (DAA) mechanism** [20]. For the description of the crypto primitives employed and actions that take place, we adopt the notation depicted in Table 6.1.

The Edge Verifiers perform one-to-many attestations with all *Prover* devices in the swarm denoted as $IoT\ Devices$.

6.1 ASSURED swarm Attestation

6.1.1 High-Level Overview of ASSURED SA

As discussed in D3.1 [18], ASSURED swarm attestation protocol focuses on the verification of the correct configuration state or behaviour profile of a swarm S comprising a tuple of algorithms $(setup, request, attest, report, verify)$. In the following, we describe each of these phases.

$setup$**:** Algorithm executed by the Vrf (SCB in our case) and all Edge Device and IoT Devices in order to deploy S into the application domain. It includes 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 (Edge) device registration and enrollment phase where it certifies the creation of the DAA Key with the Privacy CA and then also authenticates to the Blockchain CA for getting the appropriate credentials to participate in the ASSURED Blockchain infrastructure and be able to download the attestation policies to execute [28]. The latter is performed by the IoT Devices when the swarm S is constructed under a specific Edge Device: All IoT Devices create these short-term anonymous signing keys and register them to their parent Edge Device so that *linkability* can also be enabled when needed. More specifically, in ASSURED, the swarm consists of two layers: **Edge Devices and IoT Devices**. Each Edge Device is equipped with a TPM-based Wallet [27] and has a Group/DAA key. An Edge Device is a parent of a set of IoT Devices. Each IoT Device has a

[1] A detailed construction of an Enhanced swarm Attestation scheme, considering also dynamic network topology hierarchies, multiple verifying devices as well as mobility for the proving devices, will be documented in D3.7 and the second release of the ASSURED SA-DAA scheme.

pre-loaded long-term key, and it generates short-term pseudonyms certified by an Edge Device using the DAA key. All the cryptographic primitives related to the *setup* phase of both edge and IoT Devices are presented in details in Section 6.2.1.

Table 6.1: Notation used in ASSURED swarm Attestation Scheme

Symbol	Description
$G_1,\ G_2$ and G_3	Three groups with generators g_1, g_2 and g_3 respectively
e	Is a pairing function defined as $e : G_1 \times G_2 \rightarrow G_3$ and $e(g_1, g_2) \rightarrow g_3$
(x, y)	The Privacy CA private key
(X, Y)	The Privacy CA public key
E_j	The j^{th} Edge Device
tsk	E_j's private key
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 Security Context Broker (SCB)
T	The SCB's Tracing Key
(x_L, y_L)	The IoT Device long-term Key-pair
(x_p, y_p)	The IoT Device short-term Key-pair
σ^L_{DAA}	The Edge Device DAA signature on the IoT Device long-term public key y_L
σ^p_{DAA}	The Edge Device DAA signature on the IoT Device short-term public key y_p
σ_i	The i^{th} IoT Device signature on its state m_i using the short term secret key x_p
ch	Attestation challenge
ρ, r, s	Random numbers in $\mathbb{Z}_q$
v	The total numbers of Edge Devices in the swarm
k	The total numbers of IoT Devices in the swarm
P	The total numbers of certified pseudonyms by the Edge Device for each child IoT Device
$\sigma_{[1-n]}$	The aggregated signature of n IoT Devices
bsn	Edge Device random input in $\{0, 1\}^*$ used for tracing and linking two DAA signature
H	Hash function defined as $H : \{0, 1\}^* \rightarrow G_1$
π^1, π^2	Proofs of construction of Q_j and T_j respectively
π_{ipk}	Proof of the CA public key construction
π^{CA}_j	Proof of the E_j's credential construction

$request$: Algorithm initiated by Vrf to request an authenticated measurement from the swarm devices. To initialize an attestation procedure in ASSURED, the challenge ch will be retrieved through the attestation policy on the Blockchain. This challenge will first be signed and then sent to all the devices participating in the swarm.

$attest$: Integrity-ensuring algorithm implemented and executed individually by each device in response to the attestation request. This algorithm produces an attestation result m_i. Each IoT Device $i \in S$ creates such a message based on the traces extracted during run-time from the ASSURED Tracer [26]. To preserve the generality of the swarm attestation protocol in ASSURED, the attestation procedure that computes the integrity measurements is agnostic to the type of attestation used. In this way, we describe the static attestation as a basic approach, but without loss of generality, the same protocol can be extended to perform Control-Flow Attestation, Configuration Integrity Verification, etc. Once the attestation is completed, the device signs the attestation result m_i using it own short-term key (certified by its parent Edge Device). Each Edge Device,

upon reception of these signed messages (depicting the system measurements), first verify the signature and if correct they then proceed with the verification of the actual measurements as part of the overall attestation process [20]. Depending on the attestation result per each IoT Device and the required level of privacy (Section 4.2), the Edge Device will create the appropriate aggregate result and will sign it with its DAA Key. Furthermore, each Edge Device performs its own attestation using the zero knowledge proofs in the DAA scheme. Section 6.2.2 describes the cryptographic primitives of this phase.

$report$: Algorithm executed between S members to aggregate individual m_i results into the swarm attestation result, which is then delivered to the root Vrf. Once an IoT Device i signs the attestation result, it sends the signature to the Edge Device. Upon receiving the signatures from all children IoT Devices, each Edge Device verifies the signatures of their children and aggregates only the valid signatures. This means that if any IoT signature verification doesn't pass, it will not be included in the aggregated signature. Then, together with the aggregated IoT signatures, each Edge Device concatenates its group signatures created on the IoT Devices pseudonyms. After aggregating the IoT Devices signatures of different Edge Devices, the final swarm signature is sent to the Verifier. The crypto primitives related to Edge Device signatures are described in Section 6.2.2.3.

$verify$: Algorithm executed by Vrf after completion of report to check whether all devices in the swarm are in a trustworthy state. Upon receiving the attestation result from IoT Devices, each Edge Device acts as a Verifier performing the verification of the received measurements. Afterwards, the Edge Devices can follow one of these approaches: (1) the necessary actions for achieving complete privacy, (2) privacy-preserving for pseudonyms, and (3) no privacy. In addition, ASSURED enables any other stakeholder to check the result on the Blockchain by leveraging the traceability features with the Group Tracer. In particular, the tracing of IoT Device can be achieved by the Edge Device that has the knowledge of the IoT long-term key corresponding to the certified short-term pseudonyms. The crypto details of the *verification* procedure are presented in Section 6.2.3.

6.1.2 Preliminaries

For the successful setup of a swarm attestation, we assume that their are two main authorities; **a Group Issuer and a Group Tracer** which is the Security Context Broker (SCB). The main building blocks required for our construction are pseudonyms and Direct Anonymous Attestation(DAA) with both **traceability and device-controlled linkability**. **DAA Traceability is an additional property needed for our swarm attestation and achieved by giving the SCB a tracing key T to trace Edge Devices identities from their corresponding signatures**.

An Edge Device is a parent of a set of IoT Devices. Each Edge Device has a Group/DAA key and each IoT Device has a long term key, which is a pre-loaded key in each IoT Device. The public part of the long term key represents the identity of the IoT Device. Also we assume that each IoT Device creates a set of short-term pseudonyms, which are certified using a Group/DAA signature created by an Edge Device (created during the $setup$ phase).

For the creation of the aggregate swarm attestation result, we will follow the aggregate signature scheme as in [6].

- Upon registration with the ASSURED Privacy CA (as defined in D4.1 [19]) that is responsible for certifying the correct attributes needed to be produced by newly enrolled and registered devices, the IoT Devices will create their own short term keys (x_p, y_p) that will be

certified by their parent Edge Device using the edge own DAA Key; i.e. an Edge Device creates a DAA signature on the public short term key y_p. At the end of registration, the IoT Device will get a bunch of pseudonyms that are created for later execution of the protocol.

- Whenever a swarm wants to create swarm attestation, each IoT Device signs a message m using it own short term key certified by its parent Edge Device, the device computes the signature $\sigma_p = H(m)^{x_p} \ mod\ q$.
- Devices send their signatures are then sent to their parent edge that verifies the signature. An Edge Device checks whether $e(\sigma_p, g_2) = e(H(m), y_p)$ holds for each child device.
- If the verification is successful, the Edge Device aggregates its n children's signatures $\sigma_1 \ldots \sigma_n$, an edge simply computes the aggregate signature $\sigma_{[1-n]} = \sigma_1 \times \ldots \times \sigma_n = H(m_1)^{x_1} \ldots H(m_n)^{x_n}$
- Edge Device creates its DAA signature and concatenates it together with the aggregated IoT signatures.
- After aggregating the IoT Devices signatures of different Edge Devices, the final swarm signature is sent to root Verifier.
- To verify this signature, the root Verifier checks whether the following equation holds

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

  where k is the number of all IoT Devices in a swarm. The Verifier verifies all the Edge Devices DAA signatures.
- The tracing of edges is achieved by the SCB using its tracing key to trace the DAA signatures. The tracing of IoT Devices is achieved by the parent Edge Device through the certified IoT pseudonyms that are linked to their corresponding long term key in the Edge Device record.

6.2 Static swarm Attestation Protocol

In what follows, we proceed with a detailed description of all the operations and sequence of actions that take place during the instantiation of the swarm attestation protocol.

6.2.1 Setup Phase

Creating the CA keys: The Privacy CA generates her key pairs $x, y \leftarrow \mathbb{Z}_q$, sets $X = g_2^x$ and $Y = g_2^y$. The CA proves that his key is well formed and register her key (X, Y, π_{ipk}) with ASSURED authorities (Blockchain CA).

6.2.1.1 Edge Device Enrollment

Certifying Edge Device DAA Public Keys by the Privacy CA: Upon receiving a registration and enrollment request session from an Edge Device E_j for $j \in [1, v]$, where v is the total number of Edge Devices in the swarm, the Privacy CA chooses a fresh nonce ρ and sends it to the Edge Device E_j who forwards it to its TPM M_j. The TPM then chooses his secret key $tsk \leftarrow \mathbb{Z}_q$,

sets $Q_j = g_1^{tsk}$ and computes π_j^1, a proof of construction of Q_j. M_j sends (Q_j, π_j^1) to the CA via E_j. The Privacy CA then verifies π_j^1 and that the platform is eligible to join, then generates the credential for E_j as follows:

The Privacy CA chooses $r \leftarrow \mathbb{Z}_q$, sets $a = g_1^r$, $b = a^y$, $c = a^x Q_j^{rxy}$ and $d = Q_j^{ry}$, it also generates a proof π_j^{CA} on his credential constructions.

Generating the Tracing Keys by the ASSURED Security Context Broker SCB: Each TPM M_j sets $T_j = g_2^{tsk}$, and computes π_j^2, a proof of construction of T_j. M_j sends $(T_j, Q_j, \pi_j^1, \pi_j^2)$ via E_j to the SCB. The SCB verifies π_j^1 and π_j^2. Checks that

$$e(Q_j, g_2) = e(g_1, T_j)$$

is valid. To convince SCB that T_j is a right key, SCB may also be given 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, ASSURED SCB 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 $j \in [1, v]$.

6.2.1.2 IoT Device Enrollment

1. Upon registration to the traget network, each IoT Device will have a pre-loaded long term key pair (x_L, y_L) that is certified by its parent Edge Device.
2. The Edge Device certifies its children IoT long term keys by creating a DAA signature σ_{DAA}^L on y_L, the public part of the IoT long term key.
3. Each IoT Device creates short term random keys ($x_p \in \mathbb{Z}_q, y_p = g_2^{x_p}$), for $p \in [1, \ldots P]$ where P is the total number pseudonyms that will be certified by the Edge Device.
4. The IoT Device's short term keys are then certified by their parent Edge Device using the edge own DAA key; i.e. the Edge Device creates a DAA signature/pseudonym σ_{DAA}^p on each y_p.
5. The Edge Device sends $(y_p, \sigma_{DAA}^p, y_L, \sigma_{DAA}^L)$ to the child IoT Device.
6. The Edge Device keeps the certified IoT pseudonyms together with the corresponding IoT Device long term public key y_L
$$(y_p, \sigma_{DAA}^p, y_L, \sigma_{DAA}^L)$$
in its records. This record that links each set of certified IoT pseudonyms to their long term key for each child IoT Device, this record will help the Edge Device in tracing back the identity of any of its children IoT signatures.

6.2.2 Attestation and Report Phase

6.2.2.1 IoT Device Signature on the Extracted Measurement

To create swarm signatures, the Edge Device sends a challenge ch, that has been extracted by the attestation policy from the Blockchain, to the IoT Devices to initiate the swarm attestation. IoT Devices follow the following steps:

1. Whenever an IoT Device receives a request to create a swarm attestation, each IoT Device signs a message m using it own short term key that is previously certified by its parent Edge Device during enrollment phase, the IoT Device then computes its signature

$$\sigma_p = H(m)^{x_p} \ \ mod\ q$$

2. Each IoT Device send its signature σ_p together with y_p to its parent edge that verifies the signature and checks whether

$$e(\sigma_p, g_2) = e(H(m), y_p)$$

holds.

3. If the signature verification pass for the n IoT children devices, the Edge Device aggregates its n children signatures $\sigma_1, \ldots \sigma_n$.

4. The aggregate signature $\sigma_{[1-n]} = \sigma_1 \times \ldots \times \sigma_n = H(m_1)^{x_1} \ldots H(m_n)^{x_n}$.

5. If any IoT signatures verification doesn't pass, or in case of if any missing IoT Device signature, the Edge Device will not add this signature into the overall aggregated signature.

6.2.2.2 Edge (Privacy-Preserving) Verification of Measurements & Construction of Aggregate Signature

In the ASSURED framework, the aggregated attestation result should reflect the claimed integrity measurement during the attestation. In this context, Edge Devices act as Verifiers of the measurements (i.e., attestation result) reported by the IoT Devices. Upon receiving a measurement, a given Edge Device can follow one of the following approaches based on the **privacy protection profiles** defined in Section 4.2:

1. **Full Anonymity (Privacy-Preserving Mode):** ASSURED framework distributes trust among Edge Devices allowing each Edge Device to certify its children IoT pseudonyms making it infeasible for any other device (including Edge Devices and authorities) to produce IoT signatures on behalf of the IoT Devices without been certified by an Edge Device. 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 this IoT Device in the aggregation. This approach does not reveal any information about the identity of the devices included in the aggregated signatures, thus, it provides strong privacy guarantees. In order for authenticated Verifiers to be able to link a (failed) attestation result back to the IoT Device identity, they need to interact with the SCb acting as the *Tracing Entity*.

2. **Partial Anonymity (Privacy-Preservation based on Pseudonymity):** In this approach, the Edge Device will include all the IoT signatures irrespective of whether the device attestation passed or not (successful or failed attestation result). Failed attestation signatures are added in the aggregate result but not added as separate record in the ledger. In this context, the SCB (as the tracing Authority) is the only entity that when a corrupt Edge Device is detected, it will use its tracing key to extract the corrupt device identity from its DAA signature provided in the swarm attestation. In addition, Edge Devices are able to trace back the identity of the IoT Device, relying on its records that list each IoT Device certified short term keys together with its corresponding long term key.

3. **Non-Privacy-Preserving Mode:** In this approach, the aggregate signature of the swarm includes the precise list of device identifier included in the attestation and that do not pass the attestation. This information is added on the ledger allowing third-parties to gain knowledge about the corrupted devices. Thus, this approach does not provide any privacy guarantees.

6.2.2.3 Edge Device Signature & Traceability

Edge Devices create traceable DAA signatures as follows:

1. To sign a message μ with respect to a basename bsn which is decided by the Edge Device (random string in $\{0,1\}^*$) that is needed for linkability and traceability of the DAA signature, the Edge Device E_j re-randomizes its credential by choosing a random $s \leftarrow \mathbb{Z}_q$, and sets $(a', b', c', d') \leftarrow (a^s, b^s, c^s, d^s)$.

2. E_j then sends (μ, bsn, s) to the TPM M_j who checks that $b' = b^s$ and $d' = d^s$, sets $nym = (\text{bsn})^{tsk}$, and calculates a proof of knowledge, on the TPM secret key tsk and the device credential, SPK on (μ, bsn) as

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

   The DAA signature is $(a', b', c', d', SPK, nym)$.

3. The swarm signature is

$$\left(\sigma_{[1-k]}, \sigma^j_{DAA}, (\sigma_i, y_i)\right)$$

   $\forall\ i \in [1-k]$ and $j \in [1-v]$, where k and v are the the total number of IoT and Edge Devices in the swam respectively.

4. After aggregating the IoT Devices signatures of different Edge Devices, the final swarm signature will be placed on the ASSURED ledger and can be verified by any block chain user with access rights to the ledger.

6.2.3 Verification Phase

6.2.3.1 Attestation Report Verification

1. The Verifier, upon accessing an attestation report and recovering a swarm signature, verifies each Edge Device DAA signature $\sigma = (a', b', c', d', nym, SPK)$ on a message μ w.r.t. a basename bsn that is published with the signature, it verifies SPH w.r.t. (μ, bsn) and nym.

2. Checks that $a' \neq 1$, $e(a', Y) = e(b', g_2)$, $e(c', g_2) = e(a'd', X)$.

3. For revoking Edge Devices, the SCB will initiate the revocation protocol explained in Deliverable D3.2 [20] and presented in detailed in [39].

4. **IoT Device revocation check**: For all x_{Rev} in the Key Revocation List KRL saved in each Edge Device records, the Edge Device checks that $H(m_i)^{x_Rev} \neq \sigma_i$ for each child device i.

5. A Verifier checks whether the following equation holds

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

where k is the number of all IoT Devices in a swarm. In case that the aggregate result has less than k IoT aggregated signatures (**full anonymity** privacy protection), or any failed signature (**partial anonymity** privacy protection), then the attestation is failed and the Verifier can initiate the tracing of the device id for the failed attestation report.

6. It sets $f = 1$ if all the checks passed and $f = 0$ otherwise.

6.2.3.2 Link

When an external Verifier has access to different swarm attestation reports, she can check if the DAA signatures originates from the same Edge Devices or not if both signatures are signed under the same basename, which is known by the Verifier.

Roughly speaking, 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, if both signatures are valid and $nym_1 = nym_2$ output 1, otherwise 0.

6.2.3.3 Tracing/ Opening by ASSURED SCB

As aforementioned, there are two levels of traceability in our swarm attestation protocol: for tracing an aggregate attestation signature back to the Edge Device and for also tracing a failed attestation result al the way back to the origin IoT Device. We add a traceability requirement for the existing DAA scheme [15] to obtain a novel **Traceable DAA** protocol that meets our swarm attestation security and privacy requirements (Sections 4.1 and 4.2). The first level of swarm attestation traceability relies on the traceability of the DAA signature that can be done by ASSURED SCB using its tracing key. The second level relies on the static swarm topology that links each parent to their children. Tracing the IoT Device can be done by the Edge Device that has the knowledge of the IoT long term key that corresponds to the certified short term pseudonyms.

Starting with a DAA signature σ_{DAA} with a corresponding link token $nym = H(\text{bsn})^{tsk}$, the SCB uses its tracing key $T = \{T_1, T_2, \ldots, T_v\}$ to check if following equation is satisfied.:

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

If successful, then the SCB recovers that the corresponding TPM public key Q_j that corresponds to T_j from its records. If the Edge Device id is recovered, then it is easy for the Edge Device to recover any of its children long term public keys that are certified by this Edge Device relying on its records.

6.3 Amending Attestation Schemes to Swarm Attestation

Stand-alone attestation schemes such as the newly defined Control-Flow Attestation (CFA) and Configuration Integrity Verification (CIV) (described in D3.2 [20]), need to be amended to fit the ASSURED swarm Attestation protocol.

When an attestation task is executed in the context of swarm attestation, different properties need to be considered:

1. The properties of the swarm attestation;
2. The properties of the underlying attestation task.

The properties of the swarm attestation were extensively discussed in Chapter 4 whereas the properties of the other attestation schemes are documented in D3.1 [18]. The few adaptations needed, when using the attestation schemes in the context of ASSURED SA, are listed below.

In general, the attestation flow does not change. Published attestation policies [23] are received through the TPM-based wallet [29] containing both the system properties to be attested (identifier of the software function or type of loaded binary whose configuration need to be checked) and a pointer to the valid traces or hashes, depending on the attestation scheme. The traces/hashes to validate are denoted in the context of swarm attestation with m_i.

The first difference is that, instead of a single policy, for the swarm attestation a set or list of policies need to be provided, describing the different attestation tasks to be performed. After the attestation tasks were executed, the Edge Device collects the results and combines them into an aggregate signature, attesting to the trustworthiness of all evaluated entities.

The second difference is an additional parameter in the policy for the swarm attestation, which defines the procedure in case of an attestation failure or success for a device as well as the swarm attestations privacy level. As described in Section 4.2, the Verifier is required to pass the failed or succeeded attestation report to the TPM-based wallet, which in return publishes it on the block chain. In case of swarm attestation this publishing of potentially privacy eroding information may be undesirable. Depending on this parameter (which denotes the privacy protection profile to be achieved for each device in the swarm), the attestation result is published to the Ledger (Non-Privacy-Preserving Mode), successful and unsuccessful attestations are added to the group signature but not recording (Partial Anonymity) or in addition, unsuccessful attestations are omitted in the group signature, s.t., no information about the device is given, except the absence of a successful attestation result (Full Anonymity).

6.4 Research Plan for Achieving Secure & Scalable Aggregate Network Attestation (2^{nd} Release)

As described in Chapter 2, the first release of the ASSURED SA protocol considered only **static topologies with one root Verifier**, the SCB, been responsible for verifying the internal IoT Device signatures comprising the overall aggregate signature calculated by the Edge Devices. Furthermore, while it achieves all the **security requirements** and **device identity privacy** (Section 4.1), there are still additional features for enhancement towards the second release (to be documented in D3.7 scheduled for release on M30) as per the plan below:

6.4.1 Attestation Evidence Privacy

As described in Section 4.2, another important privacy aspect for consideration is 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, to overcome the restriction of current attestation solutions regarding the trustworthiness of the Verifier who needs to have full knowledge of the Prover's program execution details and in-memory layout. This renders them prone to privacy breaches and **implementation disclosure attacks** by which "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.

Towards enabling this privacy enhancement, ASSURED is already working on a new CFA design which is, to the best of our knowledge, the first privacy-preserving CFA protocol. The protocol

complements existing CFA schemes with fundamental middleware to facilitate more distributed and privacy-sensitive networks by enabling Provers to attest to a critical program's correct execution in **zero-knowledge** while enabling the entire network to benefit from the security guarantees of remotely verifiable CFA. In particular, this new enhanced protocol will embed the claim that a program's execution path is benign as a bundle of mathematical statements in an arithmetic circuit, which the Prover can prove in zero-knowledge. In this context, we will employ prominent Groth16 zkSNARK proof system and combine current advancements for optimizing **verifiable computing** circuits to maximize the efficiency of ASSURED design.

Unlike CFA schemes that consider resource-constrained provers and powerful verifiers, proof-based verifiable computation (VC) enable weak verifiers to outsource computationally-heavy computations to powerful yet untrusted provers who return proof that the computation was done correctly. Moreover, to enable secret inputs in the computations, privacy-enhanced VC schemes, e.g., zkSNARKs [38] guarantee that the proof reveals nothing about the prover's secret inputs. Furthermore, whereas proof generation is slow, verification is remarkably fast, which has made zkSNARKs attractive, especially in Distributed Ledger Technology, e.g., in the anonymous cryptocurrency Zcash [43] and adaptation in the smart-contract platform Ethereum.

Thus, ASSURED will investigate the integration of such circuits for enabling untrusted Verifiers to attest to the correct program execution of a Prover in zero-knowledge. This will achieve the required level of attestation evidence privacy in the one-to-one remote attestation paradigm. Then, this protocol will also be integrated into the many-to-many swarm attestation concept of ASSURED by also employing **ring signatures (together with the currently used group-based signatures) so as to guarantee the correct verification of an attestation result while hiding the attestation evidence as part of the swarm ring.**

Ring signatures are a type of anonymous digital signatures, where the signer takes a number of public keys, called the ring, and 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. While group signature requires an opening authority, ring signature enables a signer to generate a message anonymously in the name of others without a manager. The Verifier checks only the validity of the signature, but can not know the identity of the real signer [28]. Thus, in the context of ASSURED, such ring signatures will be employed for "*hiding*" the content of an attestation report by having the IoT Devices creating appropriate (signed) attestation messages for convincing a Verifier that their configuration and/or behavioural state is within an allowed list of correct traces (called *golden measurements*) without, however, revealing their exact state. The challenging task here is how to integrate the use of group-based signatures, for device identity privacy, with ring signatures for guaranteeing attestation evidence privacy.

6.4.2 Dynamic Topology Consideration

Another important feature that we will investigate, is to consider ASSURED SA in the context of **dynamic topologies** where the **mobility factor of devices is crucial and the verification can be supported by multiple Verifiers** (instead of having only one root Verifier as is the current case of the assumed tree-based topology structure). In such scenarios (consider for instance the case of "*Smart Manufacturing*" where robot devices can move on a manufacturing floor jumping from one IoT Gateway to another, as their parent), 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 swarm devices as they are moving**

from one swarm to another. More specifically, when an IoT Device has registered its keys and pseudonyms with an Edge Device, if during its trip, it eventually gets connected with another Edge Device, it should not be required to re-create all the cryptographic material but it should be possible for the Edge Devices to coordinate between them for securely migrating the crypto primitives from one Edge to the other Edge Device.

Towards this direction, before returing the attestation result of the current device, the Edge Device first attest all the state of its children devices and collect the statistic results of them. The statistic results contain the total number of devices attested, the number of honest devices attested, and the set of identities of dishonest devices in the subtree with this Edge Device as root. In case of mobility, if one of the leaf nodes (children device), move to another subtree then a secure exchange of this node's crypto materla will commence between the two root Edge Devices leveraging the secure *migrate functionality* of the underlying TPM-based Wallet.

Chapter 7

Security Analysis

In this last technical chapter of the deliverable, we discuss the ASSURED SA scheme with respect to the **required security and privacy properties** (Table 7.1) as defined in Chapter 4. Recall that the threat model considered was described in Section 3.1, and the endmost goal is to **formally verify the protocol against the trust model defined in D3.1** [18]. Details of the formal verification models will be included in the next version of this deliverable (D3.7). We will consider formal verification of the attestations protocols, which allow one to prove correctness using formal logic and mathematically rigorous arguments. Formal verification will be applied to top-level swarm attestations protocols, and we use a correct-by-construction methodology to develop executable implementations. Then simulation will be used to show applicability of the swarm attestations protocols across the envisioned applications.

swarm attestations Security Requirement	Description
Swarm.SC.1 - Unforgeability	It is computationally infeasible to produce (attestations) message signature pairs (M;σ) that are accepted by the verification algorithm without knowledge of the user's secret key. The unforgeability of the proposed swarm attestations protocol is built on the Unforgeability of the DAA signature protocol (Check the DAA security proof in Deliverable 3.2 [20]) combined the unforgeability of the aggregated signatures in [6]. This property can be achieved only if the Privacy CA (that performs the certification of the validity of a devices' TPM and also certifies the correctness of the produced DAA Key parameters) and Edge Device (as the data aggregator, it should be calculating the final aggregate signature correctly) are honest.
Swarm.SC.2 - Linkability of swarm attestations	The linkability of swarm attestation actions consists of two levels: The first level allows an external Verifier to link aggregate signatures (containing the overall attestation result) back to the Edge Devices, while the second level allows a mother Edge Device to link the signatures of its children back to the IoT Devices. **Linkability of Edge Devices**: The linkability of Edge Devices' signatures in a swarm attestation is based on the linkability of the DAA signature. The linkability of swarm signatures is controlled by an input parameter called the $basename$, which is a random input chosen by the Edge Device or an external Verifier. Our swarm attestation protocol supports user-controlled linkability which means that two signatures created by the same Edge Device may or may not be linked from a Verifier's point of view. If an Edge Device chooses to sign twice under the same $basename$ then signatures are linkable, it means that any external Verifier with access rights to two distinct swarm attestationss that are signed under the same $basename$ can check if the Edge Device signatures originate from the same device or not but without disclosing the Edge Devices' ID.

	Linkability of IoT Devices: The linkability of the IoT Devices in a swarm attestation relies on the static topology of the swarm. As was described in Chapter 6, each parent Edge Device has in its records the set of all certified pseudonyms associated with the corresponding IoT Device long-term key (used as identity of the IoT Device) for each child IoT Device. Thus, whenever an IoT Device creates two signatures, in two different SA instantiations, the Edge Device is able to identify, relying on its records, if these two signatures originate from the same IoT Device or not (Edge Device uses traceability to support linkability of IoT Devices). This "*pseudonym resolution*" process is invoked only in the case of a failed attestation report when an external Verifier needs to be able to trace back to the identity of the IoT Device for which this failure is an indication of risk. In this case, first, the link back the Edge Device needs to be performed (through the SCB that leverages its *tracing key*) and then the Edge Device is asked to resolve the identity of the IoT Device based on the used pseudonym a part of the aggregate signature.
Swarm.SC.3 - Anonymity	Our swarm attestation protocol supports full privacy against external Verifiers that might be honest but curious to de-anonymize all devices even if they are at a correct state (no failed attestation). *Anonymity is achieved only if the majority of devices, in a swarm, are honest - otherwise, if they release the pseudonyms they used for participating in the SA, then an eavesdropper can link back the other pseudonyms to the remaining devices*. The level of privacy for the IoT Devices is directly linked to the **group dynamicity** (Section 5.2.1). We will explain the privacy of our swarm attestations both for the level of Edge and IoT Devices: **Privacy of Edge Devices**: An external Verifier that is given two swarm signatures with different tags (signatures $basename$s using the Edge Device secret DAA key), cannot distinguish whether both signatures were created by one honest Edge Device, or whether two different honest Edge Devices created the signatures. Anonymity holds even if these two signatures are given back to the Privacy CA since this authority can identify whether these signatures originate from a valid TPM but cannot identity which TPM Endorsement Key was used. **The identity of each Edge Device will be actually protected/hided by performing the attestations using the zero knowledge proofs in Direct Anonymous attestations scheme**. **Privacy of IoT Devices**: This is only achieved against external Verifiers. In our proposed SA protocol, IoT Devices generate fully **random** pseudonyms that don't reveal the IoT Device identities. Thus, starting with an IoT pseudonym, an external Verifier cannot tell the identity of the IoT Device that generated this pseudonym unless it collaborates with the Edge Device.
Swarm.SC.4 - Traceability	Recall that the requirement is that a swarm signature (integrated in the overall aggregate signature) should be traceable to the swarm member who issued it, in the case of a failed attestation report. We consider two levels of traceability depending on whether the swarm member is an Edge Device or an IoT Device. **Traceability of Edge Devices**: In the ASSURED framework, we assume that the only authority that can recover the Edge Device's identity is the Tracing Authority which, in our context, is the ASSURED Security Context Broker (SCB). If a corrupt Edge Device is detected, the SCB will use its tracing key to extract the corrupt device identity from its DAA signature provided in the aggregate signature - more details are presented in the Tracing phase of the swarm attestation protocol description in Section 6.2.3.3). **This traceability property is a crucial security requirement for ASSURED designed SA that is added to the DAA protocol to achieve a Traceable DAA protocol which is the main ASSURED building block for constructing an efficient privacy preserving swarm attestations scheme**.

	Traceability of IoT Devices Whenever an IoT Device generates a signature, the parent Edge Device will be able to trace back the identity of the IoT Device, relying on its records that list each IoT Device certified short-term keys together with its corresponding long-term key. We should highlight that if an IoT Device chooses to create its signatures using a non certified short-term key, then the Edge Device would not aggregate this IoT signature in the overall SA aggregate result.
Swarm.SC.5 - Non-frameability	No adversary can create swarm signatures on a message that links to an honest Edge Device when this honest Edge Device never signed it. This property should hold even if Privacy CA is not honest. The non-frameability of our swarm attestation protocol is based on the non-frameability of the DAA scheme in [15].
Swarm.SC.6 - Data Integrity and Authenticity Guarantees	The aggregated attestation result should reflect the claimed (system) integrity measurement extracted for the underlying attestation task (e.g., CFA or CIV). This requirement is full-filled by the offering of **Hash Functions** to be able to guarantee the integrity of the data, and by the availability of the **Traceable Direct Anonymous Attestation)** to allow the different data sources to be authenticated and provide the necessary guarantees to the other entities which are requested to perform the different activities foreseen over the ASSURED infrastructure. In our swarm attestation, the integrity measurement during the attestation is preserved using a collision resistant one-way hash function H for creating the IoT and Edge Devices signatures.
Swarm.SC.7 - Provenance	A Verifier (Vrf) of a swarm can verify that the aggregate result is based on the target devices, comprising the swarm of interest, by certifying the identity of each device without though been able to link parts of the signature to the original device creators.
Swarm.SC.8 - Coalition resistance	This pertains to the possibility of a group of signers, in a swarm, colluding together to generate signatures that cannot be traced back to any of them. Our swarm attestation protocol achieves coalition resistance based on the following features: 1. Our protocol requires that the ASSURED SCB keeps updating its tracing key whenever a new device joins the swarm; 2. The Edge Device tracing key is directly linked to its DAA public key generated in the enrollment phase and certified by the Blockchain CA. Thus, no device can obtain a tracing key without being registered with the Privacy CA [28]. Consider the following scenario where a colluding Edge Device creates a public key Q_c with a corresponding tracing key T_c in order to create a signature that doesn't trace back to it, since Q_c is not a certified key and, hence, has no valid credential from the Privacy CA. In this case, the proof of knowledge SPK that is a part of the DAA signature cannot be generated due to the soundness of the zero knowledge proofs, and hence the signature will not pass the verification phase. Therefore, it is infeasible for colluding members to create a signature that cannot trace back to any of them since the DAA proof of knowledge SPK cannot be constructed unless the signature is created by a certified DAA key that links to a certified tracing key that is included in the SCB tracing key.
Swarm.SC.9 - Exculpability	This is the property that no member of the group, and not even the Privacy CA or ASSURED SCB, can produce swarm attestation signatures on behalf of other swarm members. This property is achieved in our proposed scheme, since we distribute the trust among different ASSURED entities, namely the Privacy CA, ASSURED SCB and the Edge Devices. Giving trust on the Edge Devices' side to certify its children IoT pseudonyms makes it infeasible for any swarm member, even authorities, to produce IoT signatures on behalf of the IoT Devices without getting certified by an Edge Device. Also, due to the unforgeability of the DAA signature, no swarm member can sign on behalf of any Edge Device without knowing the corresponding edge secrete key tsk.

Swarm.SC.10 - User Revocation	We consider two kinds of revocations depending on whether a device is an Edge or IoT Device: **Edge Device Revocation**: In ASSURED, we envision that the Edge Device, upon registration with ASSURED Blockchain CA, will create its own pseudonyms that are used to self-certify its credential and this certification can be verified by any Blockchain user (equipped with a TPM-based Blockchain Wallet). This is called pseudonymity which is the ability of an Edge Device to use a resource or service without disclosing the user's identity while still being accountable for that action. Any Revocation Authority (RA), which may also be the SCB, can then remove misbehaving Edge Devices without revealing the Edge Device's identity [20]. The intuition behind the revocation scheme in ASSURED, is to be able to de-activate any credentials and short-term signature keys, of an Edge Device, created during their enrollment in the overall IoT ecosystem: essentially, revoking the use of any Attestation Key (AK), created during the Edge Device enrollment phase. **IoT Device Revocation**: Our swarm protocol supports revocation of the IoT Devices by their parent Edge Device. We assume that edge Edge Device will have access to a set of IoT revoked keys called the Key Revocation List KRL, the Edge Device checks that each of its children IoT signatures were not produced by any key in KRL. Relying on the trust of the Edge Devices and the accuracy of the updated KRL, Edge Devices would also be able to correctly revoke the IoT Devices that their keys are KRL.

Table 7.1: High-level Security Analysis of the ASSURED swarm attestations Scheme

Chapter 8

Conclusion

This document presents the first release of the ASSURED Secure and Scalable Aggregate Network Attestation towards establishing **attestable trust to groups of devices in large-scale dynamic networks**. Towards this direction, properties of swarm attestation were introduced and how our approach copes with different requirements of the swarm and the devices itself due to the complexity of the target ecosystems. Building on top of previously specified attestation and verification mechanisms, a new adversarial model and security requirements were detailed. Finally, this document introduces the novel Swarm Attestation scheme based on the use of grou signatures supported by an enhanced version of the DAA protocol that also enables linkability and traceability of a specific device in the case of a failed attestation result (*Traceable DAA*). All cryptographic building blocks needed for the novel signature scheme were presented and amendments to the overall ASSURED framework were also highlighted in order to make previously designed verification and attestation schemes compatible. Finally, an initial qualitative security analysis was performed on how this protocol achieves the trust requirements and models defined in Chapter 4 which serves as the predecessor for a full blown formal verification that will be documented in the second version of the deliverable.

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] Mihir Bellare, Haixia Shi, and Chong Zhang. Foundations of group signatures: The case of dynamic groups. In *Cryptographers' Track at the RSA Conference*, pages 136–153. Springer, 2005.

[2] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Eran Tromer, and Madars Virza. Snarks for c: Verifying program executions succinctly and in zero knowledge. In *Annual cryptology conference*, pages 90–108. Springer, 2013.

[3] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Succinct non-interactive zero knowledge for a von neumann architecture. In *23rd {USENIX} Security Symposium ({USENIX} Security 14)*, pages 781–796, 2014.

[4] D. Boneh, X. Boyen, and H. Shacham. Short group signatures. In *Int. Cryptology Conference (CRYPTO)*, 2004.

[5] Dan Boneh. *Aggregate Signatures*, pages 27–27. Springer US, Boston, MA, 2011.

[6] Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In *International conference on the theory and applications of cryptographic techniques*, pages 416–432, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[7] Dan Boneh and Ben Lynn. A survey of two signature aggregation techniques. 2003.

[8] Jonathan Bootle, Andrea Cerulli, Pyrros Chaidos, Essam Ghadafi, and Jens Groth. Foundations of fully dynamic group signatures. *Journal of Cryptology*, 33(4):1822–1870, 2020.

[9] Ernie Brickell, Jan Camenisch, and Liqun Chen. Direct anonymous attestation. In *Proceedings of the 11th ACM Conference on Computer and Communications Security*, CCS '04, pages 132–145, New York, NY, USA, 2004. ACM.

[10] Ernie Brickell, Liqun Chen, and Jiangtao Li. A new direct anonymous attestation scheme from bilinear maps. In *International Conference on Trusted Computing*, pages 166–178. Springer, 2008.

[11] Ernie Brickell, Liqun Chen, and Jiangtao Li. Simplified security notions of direct anonymous attestation and a concrete scheme from pairings. *International journal of information security*, 8(5):315–330, 2009.

[12] Ernie Brickell and Jiangtao Li. A pairing-based daa scheme further reducing tpm resources. In *Proceedings of the 3rd International Conference on Trust and Trustworthy Computing*, TRUST'10, pages 181–195, Berlin, Heidelberg, 2010. Springer-Verlag.

[13] J. Camenisch and J. Groth. Group signatures: Better efficiency and new theoretical aspects. In *Security in Communication Networks*, pages 120–133. Springer, 2005.

[14] Jan Camenisch. Efficient and generalized group signatures. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 465–479. Springer, 1997.

[15] 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.

[16] David Chaum and Eugène Van Heyst. Group signatures. In *Workshop on the Theory and Application of of Cryptographic Techniques*, pages 257–265. Springer, 1991.

[17] Liqun Chen and Jiangtao Li. Flexible and scalable digital signatures in tpm 2.0. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, CCS '13, pages 37–48, New York, NY, USA, 2013. ACM.

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

[19] The ASSURED Consortium. Assured blockchain architecture. Deliverable D4.1, November 2021.

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

[21] The ASSURED Consortium. Assured use cases & security requirements. Deliverable D1.1, February 2021.

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

[23] The ASSURED Consortium. Policy modelling & cybersecurity, privacy and trust constraints. Deliverable D2.2, November 2021.

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

[25] The ASSURED Consortium. Technical integration points, apis specification and testing plan. Deliverable D5.1, November 2021.

[26] The ASSURED Consortium. Assured real-time monitoring & tracing functionalities. Deliverable D3.6, February 2022.

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

[28] The ASSURED Consortium. Assured secure distributed ledger maintenance & data management. Deliverable D4.2, February 2022.

[29] The ASSURED Consortium. Assured tc-based functionalities. Deliverable D4.5, February 2022.

[30] The ASSURED Consortium. Evaluation framework and demonstrators planning. Deliverable 6.1, February 2022.

[31] The ASSURED Consortium. Security context broker specification and smart contract definition & implementation for policy enforcement. Deliverable D2.2, February 2022.

[32] Jean-Sébastien Coron. Security proof for partial-domain hash signature schemes. In Moti Yung, editor, *Advances in Cryptology — CRYPTO 2002*, pages 613–626, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

[33] Heini Bergsson Debes and Thanassis Giannetsos. Segregating keys from noncense: Timely exfil of ephemeral keys from embedded systems. In *2021 17th International Conference on Distributed Computing in Sensor Systems (DCOSS)*, pages 92–101, 2021.

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

[35] Georgios Fotiadis, José Moreira, Thanassis Giannetsos, Liqun Chen, Peter B. Rønne, Mark D. Ryan, and Peter Y. A. Ryan. Root-of-trust abstractions for symbolic analysis: Application to attestation protocols. In Rodrigo Roman and Jianying Zhou, editors, *Security and Trust Management*, pages 163–184, Cham, 2021. Springer International Publishing.

[36] Thanassis Giannetsos and Tassos Dimitriou. Spy-sense: Spyware tool for executing stealthy exploits against sensor networks. In *Proceedings of the 2Nd ACM Workshop on Hot Topics on Wireless Network Security and Privacy*, HotWiSec '13, pages 7–12, 2013.

[37] Thanassis Giannetsos, Tassos Dimitriou, Ioannis Krontiris, and Neeli R. Prasad. Arbitrary Code Injection through Self-propagating Worms in Von Neumann Architecture Devices. *The Computer Journal*, 53(10):1576–1593, 02 2010.

[38] Jens Groth. Short pairing-based non-interactive zero-knowledge arguments. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 321–340. Springer, 2010.

[39] 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, page 48–59, New York, NY, USA, 2021.

[40] Benoît Libert, San Ling, Fabrice Mouhartem, Khoa Nguyen, and Huaxiong Wang. Signature schemes with efficient protocols and dynamic group signatures from lattice assumptions. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology – ASIACRYPT 2016*, pages 373–403, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

[41] San Ling, Khoa Nguyen, and Huaxiong Wang. Group signatures from lattices: simpler, tighter, shorter, ring-based. In *IACR International Workshop on Public Key Cryptography*, pages 427–449. Springer, 2015.

[42] Anna Lysyanskaya, Silvio Micali, Leonid Reyzin, and Hovav Shacham. Sequential aggregate signatures from trapdoor permutations. In Christian Cachin and Jan L. Camenisch, editors, *Advances in Cryptology - EUROCRYPT 2004*, pages 74–90, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[43] Eli Ben Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy*, pages 459–474. IEEE, 2014.

[44] Sebastian Surminski, Christian Niesler, Ferdinand Brasser, Lucas Davi, and Ahmad-Reza Sadeghi. Realswatt: Remote software-based attestation for embedded devices under real-time constraints. CCS '21, page 2890–2905, New York, NY, USA, 2021. Association for Computing Machinery.

[45] Giannetsos Thanassis, Dimitriou Tassos, and Prasad Neeli R. Weaponizing wireless networks: An attack tool for launching attacks against sensor networks. In *Black Hat Europe 2010*, Barcelona, Spain, April 12-15, 2010.

[46] Florian Tramer, Fan Zhang, Huang Lin, Jean-Pierre Hubaux, Ari Juels, and Elaine Shi. Sealed-glass proofs: Using transparent enclaves to prove and sell knowledge. In *2017 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 19–34. IEEE, 2017.

[47] Riad S Wahby, Srinath Setty, Max Howald, Zuocheng Ren, Andrew J Blumberg, and Michael Walfish. Efficient RAM and control flow in verifiable outsourced computation. *Cryptology ePrint Archive*, 2014.

[48] Howard Wu, Wenting Zheng, Alessandro Chiesa, Raluca Ada Popa, and Ion Stoica. {DIZK}: A distributed zero knowledge proof system. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*, pages 675–692, 2018.