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

D4.6 ASSURED TC-BASED FUNCTIONALITIES – VERSION 2

Revision: v.1.0

Work package	WP 4
Task	Task 4.4
Deliverable lead	SURREY
Version	1.00
Editors	Nada El Kassem (SURREY)
Reviewers	Liquin Chen (SURREY), Thanassis Giannetsos (UBITECH)
Abstract	Deliverable D4.6 puts forth the final version of the ASSURED TPM-based Wallet, which is responsible for the decentralized identity management of all devices wishing to interact with the ASSURED Blockchain infrastructure. This deliverable is the first of its kind to present a Wallet following the methodology of Self-Sovereign Identities (SSI), which facilitates hardware-based keys to provide privacy protection to devices with a high level of assurance. The ASSURED TPM-based Wallet is aligned with the latest EU standards, and is able to manage Verifiable Credentials (VCs) and Verifiable Presentations (VPs) towards achieving the property of selective disclosure. In addition, we provide details on the Attribute-Based Encryption (ABE) scheme, which is able to encrypt attestation evidence or other safety-critical data with a high level of granularity with regards to data access, so that only devices that can exhibit specific properties are able to perform encrypt or decrypt operations. The designed scheme is the first of its kind to provide decentralized ABE, where devices are able to create encryption and decryption keys through their Trusted Computing Base (TCB), thus removing the need for a centralized entity as the master key authority. A rigorous security analysis and experimental performance evaluation are also provided for all the designed schemes.
Keywords	Self-Sovereign Identity, Authentication, Authorization, Identity Management, Wallet

Funded by the Horizon 2020 Framework Programme of the European Union

Document Revision History

Version	Date	Description of change	List of contributors
v0.1	04.01.2023	Table of Contents provided, summarizing the components and functionalities of ASSURED to be covered by the deliverable	Nada El Kassem (SURREY)
v0.2	16.01.2023	Summary of updates on the ASSURED TC-based crypto enablers for secure data management, and definition of the security properties to be fulfilled by the final version of the ASSURED TPM-based Wallet and the ABE scheme. (Chapters 2 and 3)	Nada El Kassem (SURREY) Edlira Dushku, Benjamin Larsen (DTU) Dimitris Karras, Alexandros Sampanis, Stefanos Vasileiadis, Nikos Chatzivasileiadis, Nikos Kalantzis (UBITECH) Sotiris Kousouris, Stefanos Venios (S5) Meni Orenbach (NVIDIA)
v0.3	24.01.2023	Definition of the architectural description of the TPM-based Wallet, as well as architectural details on the Wallet design, such as the issuance of Verifiable Credentials (VCs) and creation and verification of Verifiable Presentations (VPs). (Chapter 4)	Nada El Kassem (SURREY) Edlira Dushku, Benjamin Larsen (DTU) Meni Orenbach (NVIDIA)
v0.4	30.01.2023	Definition of the functional specifications regarding data management for the envisioned use cases, as well as the conceptual overview of the ABE protocol. (Chapter 5)	Dimitris Karras, Alexandros Sampanis, Stefanos Vasileiadis, Nikos Chatzivasileiadis, Nikos Kalantzis (UBITECH)
v0.5	09.02.2023	Definition of the security model and ideal functionality algorithms for the TPM-based Wallet, as well as the security proof based on the Universal Composability (UC) model. (Chapter 4)	Nada El Kassem (SURREY) Edlira Dushku, Benjamin Larsen (DTU)
v0.6	17.02.2023	Detailed description of the architectural details and building blocks of the ABE scheme for each of the comprising phases (Setup, Encrypt, Decrypt, Attribute List Update) (Chapter 5)	Alexandros Sampanis, Stefanos Vasileiadis, Nikos Chatzivasileiadis, Nikos Kalantzis (UBITECH)
v0.7	27.02.2023	Definition of the security model and description of the security proof for the ABE scheme (Chapter 5)	Nada El Kassem (SURREY) Edlira Dushku, Benjamin Larsen (DTU)
v0.75	03.03.2023	Review of the TPM-based Wallet and ABE scheme	Liquin Chen (SURREY) Thanassis Giannetsos (UBITECH)
v0.8	08.03.2023	Updates based on review to clarify action flows in methods related to TPM-based Wallet and ABE scheme (Chapters 4 and 5)	Nada El Kassem (SURREY) Edlira Dushku, Benjamin Larsen (DTU) Alexandros Sampanis, Stefanos Vasileiadis, Nikos Chatzivasileiadis, Nikos Kalantzis (UBITECH)
v0.85	21.03.2023	Performance analysis and evaluation for VC issuance, VP creation and verification, and the encryption and decryption process using ABE (Chapter 6)	Nada El Kassem (SURREY) Edlira Dushku, Benjamin Larsen (DTU) Alexandros Sampanis, Stefanos Vasileiadis, Nikos Chatzivasileiadis, Nikos Kalantzis (UBITECH)
v0.9	27.03.2023	Finalization of the introduction and conclusions section of the deliverable (Chapters 1 and 7)	Dimitris Karras (UBITECH) Nada El Kassem (SURREY)
v0.95	30.03.2023	Review of the entire deliverable	Liquin Chen (SURREY) Thanassis Giannetsos (UBITECH)
v1.0	04.04.2023	Update of the deliverable based on the review comments and submission	Jean-Baptiste Milon (MARTEL), Thanassis Giannetsos, Dimitris Karras (UBITECH)

Funded by the Horizon 2020 Framework Programme of the European Union

Editor

Nada El Kassem (SURREY)

Contributors (ordered according to beneficiary numbers)

Edlira Dushku, Benjamin Larsen (DTU)

Liqun Chen, Nada El Kassem (SURREY)

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

Sotiris Kousouris, Stefanos Venios (SUITE5)

Meni Orenbach (NVIDIA)

Funded by the Horizon 2020
Framework Programme of the European Union

DISCLAIMER

The information, documentation and figures available in this deliverable are written by the "Future Proofing of ICT Trust Chains: Sustainable Operational Assurance and Verification Remote Guards for Systems-of-Systems Security and Privacy" (ASSURED) project's consortium under EC grant agreement 952697 and do not necessarily reflect the views of the European Commission.

The European Commission is not liable for any use that may be made of the information contained herein.

COPYRIGHT NOTICE

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

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

Funded by the Horizon 2020 Framework Programme of the European Union

Executive Summary

Towards creating trust in data management transactions between devices comprising large-scale **Systems-of-Systems (SoS)** characterized by various security and privacy requirements, one of the core tenets of ASSURED is the management of the **identities of the users and devices** participating in the ASSURED framework, when handling operational- and attestation-related data when shared with other devices and stakeholders. To this end, ASSURED implements a **decentralized architecture for digital identity management** following the **Self-Sovereign Identity (SSI)** concept. The motivation behind the adoption of SSI is to enable secure identity management to devices interacting with the **ASSURED Blockchain Infrastructure**, in the context of obtaining access to stored data, such as **attestation reports** and **operational data**. SSI aims to provide devices with control over their own identity, by enabling them to only disclose their attributes that are required in order to access a particular set of data or Blockchain-related service (**selective disclosure**) through the creation and usage of **Decentralized Identifiers (DIDs)**.

To this end, we have designed and implemented the **ASSURED TPM-based Wallet**, which enables the management of all necessary cryptographic material to support a wide range of **secure and lightweight on-chain interactions**, such as **reading data**, **reading attestation policies** deployed on the ledger, **recording data** on the ledger, **recording attestation results**, and **performing queries on stored data**. The TPM-based Wallet also aims to accomplish the **data sharing agreements and behaviors** defined in D1.4 [8] that aim to fulfill the security and privacy requirements of systems and organizations utilizing the ASSURED framework. The first version of the TPM-based Wallet was presented in the first version of this deliverable, D4.5 [13], and in this deliverable we finalize the design and implementation of all the functionalities of the Wallet.

One of the main functionalities of the TPM-based Wallet presented in this deliverable is the issuance and storage of the **device attributes** in a way that achieves the aforementioned principle of selective disclosure. Specifically, the TPM-based Wallet of a device is able to communicate with the **Privacy Certification Authority (CA)** to issue the **Verifiable Credentials (VCs)** of the device, which essentially contain the entire set of attributes of the device and are stored in its TPM-based Wallet. Afterwards the TPM-based Wallet enables the self-issuance of **Verifiable Presentations (VPs)**, which contain a subset of the the attributes included in the VCs and act as DIDs that provide verifiable claims regarding the attributes and the identity of a device, when accessing a set of data or a Blockchain-related service. We also present the capabilities of the TPM-based Wallet for the **storage and management of the cryptographic material** such as the DAA Key and the Attestation Key. One additional feature of the ASSURED TPM-based Wallet is this regard is the integration of **key restriction usage policies**, which bind the usage of the key to a device state that is known to be correct and trustworthy, to ensure that the ASSURED cryptographic enablers are not used by a device that has been compromised by a malicious party.

As aforementioned, the ASSURED TPM-based Wallet enables the devices participating in the framework to participate in the ASSURED cryptographic enablers that dictate interactions with the Blockchain infrastructure and the data stored on the ledger. To this end, in D4.5 [13] we have presented the **Attribute-Based Access Control (ABAC)** scheme, which provides access to a data set stored on the Blockchain to devices that can verifiably demonstrate a VP containing the appropriate attributes. The **Dynamic Symmetric Searchable Encryption (DSSE)** scheme, initially presented in D4.3 [10], enables keyword-based querying on encrypted data on the Blockchain, without decrypting the data themselves. Finally, the **Attribute-Based Encryption (ABE)** scheme, initially presented in D4.2 [12], enables a device to encrypt a set of data in a manner that only allows a device that can create a VP with the appropriate attributes.

In this deliverable, we focus on the ABE scheme, and we provide the final version whose implementation and integration will be completed during the second reporting period. The final version of ABE provides privacy-preserving properties to the encryption process, ensuring that the encrypted data does not divulge any personally identifiable information on the encryptor device, and does not enable a malicious party to deduce implementation details of the device. In addition, we have implemented an **attribute list correctness verification** method, in order to ensure that a malicious party does not attempt to bypass the encryption policy (e.g., by using an outdated version of the policy). Finally, one important new feature of the ABE scheme is the ability to perform an **internal integrity check** of both the encryptor and decryptor devices, in order to verify that the ABE scheme is not leveraged by malicious parties in order to perform attacks.

In Chapters 4 and 5, we provide detailed descriptions of the final versions of the TPM-based Wallet and the ABE scheme, respectively. We also provide rigorous mathematical proofs of the correctness of these schemes, in order to verify the achievement of the security and privacy requirements set forth in Chapter 3. Finally, in Chapter 6, we present evaluation results for the aforementioned schemes, demonstrating their performance efficiency in practical systems.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 The Need for Zero Trust as an Enabler to Data Management Challenges

The core overarching vision of ASSURED is the provision of **operational assurance to large-scale heterogeneous Systems-of-Systems (SoS)** throughout their operational lifecycle. These may belong to various different operational domains, and may be characterized by various **security and privacy requirements**. We attempt to capture a wide range of such requirements through the envisioned ASSURED use case demonstrators, which were presented in D1.1 [9], and include *BIBA Smart Manufacturing*, *DAEM Smart Cities*, *UTRC Smart Aerospace*, and *SPH Smart Satellites.*A core aspect that needs to be considered in all these use case scenarios is the **handling of complex and sensitive data in complex operational environments**, in a manner that is able to capture all the aforementioned requirements.

Towards fulfilling the underlying requirements regarding handling and accessing such data, in ASSURED we employ the notion of **zero trust**. This means that ***we cannot make any assumptions on the trustworthiness of the actors participating in a transaction***, referring both to the parties requesting to access a set of data, and the parties responsible for the verification of the requesting devices. With regards to the former, when a device presents its credentials in order to access a particular set of data, it must also be able to provide evidence in order to bootstrap trust. For the latter, when someone aims to access a set of data by disclosing a set of attributes, we cannot assume that the party responsible for the verification of these attributes is trustworthy. This differs from the traditional notion of **Public Key Infrastructure (PKI)**, where we assume that a trusted third party is responsible for the issuance of the device credentials, and a trusted entity is responsible for allowing device interactions with the Blockchain.

In ASSURED, the handling of sensitive data pertains to their **storage on the Blockchain ledger in a secure and privacy-preserving manner**, as well as the **capability to provide access to the stored data to parties with the appropriate privileges**. Specifically, the data that needs to be handled in the context of ASSURED can be split into two categories:

- ***Operational data:*** This includes data that captures information on the internal functions and processes of a system and the devices operating within the system. For example, in the *DAEM Smart Cities* use case, this may include sensor data originating from smoke detection sensors, or video feeds originating from surveillance cameras. This data is used in order to perform the intended functionalities of the system (in the aforementioned use case, pertaining to the achievement of public safety).

- ***Attestation data:*** This includes data that is used in the context of the ASSURED attestation enablers, and is provided by Prover devices as evidence to a Verifier device, in order to prove the correctness of their configuration state, or the correctness of a control flow from the execution of a software process. For example, in the *UTRC Smart Aerospace* use case, the Ground Station Server (GSS) aims to deploy secure updates to the Secure Server Router (SSR) of an aircraft. In this case, the aircraft needs to provide evidence of the correctness of its configuration state before and after the secure update (Configuration Integrity Verification, CIV), as well as evidence of the correctness of the update process itself (Control Flow Attestation, CFA). This data is stored on the Blockchain in the form of **attestation reports**, and essentially constitutes **threat intelligence data** that can be accessed by authorized parties to obtain information regarding identified threats and vulnerabilities.

From all the above, it follows that the complex environments (SoS and supply chains) considered in ASSURED involve **large amounts of data**, which should be handled in a manner that guarantees the correct operation of the system, as well as the secure and privacy-preserving sharing of such data. This amount of data is a double-edged sword. On the one hand, it enables the **support and provision of better data sharing services**, but on the other hand it **increases the security and privacy requirements to avoid breaching the security profile of the devices**. These requirements refer to data management throughout the operational lifecycle, from the **creation** to the **storage and sharing of the data** through the Blockchain infrastructure.

In order to achieve these requirements with regards to access control for the aforementioned kinds of data, simply granting explicit trust to users and devices is not sufficient. We also need to be able to provide **protocols with different levels of assurance pertaining to data management and access**, as well as to **enable different levels of granularity in the access control in order to capture different roles of users of devices that need to showcase the necessary attributes and privileges for accessing the data**. In addition, we need to enable the aforementioned new vision of the zero trust paradigm, which makes no assumptions on the level of trust of the participating entities. This is in line with the prevailing consensus in the community of moving away from centralized PKI-based architectures that are limited with regards to scalability, and **shifting trust from the backend to the edge**. Therefore, we aim to **bootstrap vertical trust between the interacting entities**.

To this end, we need to equip each device participating in a data handling transaction with the capability to **provide evidence on its trustworthiness level**, or **provide evidence regarding the required attributes in order to participate in the transaction**. To this end, in ASSURED, **data access is determined based on the attributes that a device can exhibit in a verifiable manner**. These attributes may be related to the **identity of the device** or the **role of the device**. For example, in the Smart Cities use case, a specific set of operational data may only be accessible by police officers or firefighters, so anyone aiming to access this data must be able to exhibit the attributes that verify their identity and role.

Therefore, in order to preserve the privacy of the device identity and data, **a device should be able to only disclose the particular set of attributes required in order to access the desired set of data (selective disclosure)**. To this end, the **ASSURED Secure Device Enrollment** process (described in D4.5 [13]) includes the issuance of **Verifiable Credentials (VCs)** by the Privacy CA, which contain the set of attributes of the device and are stored in the Trusted Component (TC) of the device. Afterwards, when aiming to access a set of data that has been encrypted based on a set of attributes, the device is able to create a **Verifiable Presentation (VP)** as a subset of the issued VCs, only containing the required attributes. This approach provides

the basis for the implementation of all the ASSURED data access and management methods and algorithms, and serves towards the core tenet of ASSURED towards **shifting trust to the "edge" and providing more control to the device with regards to the management of its own identity**.

Towards this direction, in order to enable identity management of the devices in the context of ASSURED, we employ the notion of **Self-Sovereign Identities (SSI)**, as well as the **W3C structure of VCs and VPs** in order to achieve the underlying requirements. This is instantiated in ASSURED through the use of a **TPM-based Wallet**, which is protected by **HW-based keys** by leveraging the **HW-based TC** embedded in each device in order to enable it to provide verifiable claims regarding its own identity. Note that, in the ASSURED implementation, we use **Trusted Platform Modules (TPMs)** as TCs for the creation of the wallet, but the designed methods are agnostic to the type of TC, and can be adapted to any type of considered TC. The first implementation of the TPM-based Wallet was provided in D4.5 [13], where we provided details regarding the identity and data management capabilities provided by this Wallet. Note that **ASSURED is the first project of its kind to have designed an identity management wallet capable of managing VCs and self-issued VPs for selective disclosure of attributes, with a high level of assurance protected through HW-based keys**.

In this deliverable, we expand on this design and we provide an updated version of the TPM-based Wallet that addresses further data management concerns. Specifically, we aim to answer the question: *What is further needed for SSI to be able to give individuals full control and autonomy on their digital identities while establishing trust with an interaction?* With more and more services occurring online, consumers are seeking better and more trustworthy privacy controls. ASSURED's work on federated identities managed by Wallets offering high level of security and privacy assurances through the use of HW-based keys. ASSURED novel design based on the W3C Decentralized identifiers and Verifiable Credentials, enabling secure selective disclosure of only those user attributes needed for accessing a specific service, can offer a significant milestone to the privacy challenges of the current fragmented identity ecosystem comprising multiple third-party identity providers that lack interoperability.

1.1.1 Leveraging ASSURED HW-based Trust Enablers for Secure Data Management & Identity Management

In the remainder of this deliverable, we present the final version of the crypto primitives that have been designed both in the context of **access control**, as well as for guaranteeing the **integrity and confidentiality** of the shared data. With regards to the former we present the final version of the **TPM-based Wallet** that has been designed in ASSURED, which essentially showcases how **HW-based keys can achieve the envisioned property of higher levels of assurance for credential management**, while also **enabling privacy preservation via selective disclosure**. With regards to the latter, we present the design and implementation of the final version of the **Attribute-Based Encryption (ABE)** scheme, that enables the devices to encrypt devices in a verifiable attribute-based manner. The design, implementation analysis, and formal security analysis of both schemes is provided in this deliverable. These constitute the final versions of the lightweight cryptographic primitives that ASSURED offers towards enabling secure data management in the context of resource-constrained embedded systems.

In Table 1.1, we provide a high-level overview of the functional specifications of the TPM-based Wallet and the ABE scheme.

Functional Specification	Instantiation in ASSURED
The Wallet must support fine-grained **access control** and **granular usage policies** for both operational and threat intelligence data.	The deployment and enforcement of policies as smart contracts, supported by the TPM-based Wallet and the Blockchain infrastructure, is outlined in D2.6 [16] and D4.4 [1].
The Wallet must be able to provide **secure storage** and **secure management** of all cryptographic secret material and keys used in the context of the ASSURED lightweight crypto enablers.	The use of the TPM-based Wallet for secure storage is analyzed in Section 4.4.
The Wallet should be able to provide decentralized, trustworthy management of **Verifiable Credentials (VCs)** and **Verifiable Presentations (VPs)** with additional guarantees on the level of trust of the device hosting the Wallet.	The issuance of VCs and VPs using the TPM-based Wallet is analyzed in Sections 4.3.2 and 4.3.3.
The Wallet must be able to provide the **secure management of the policies** representing the set of attributes that need to be preset for being able to encrypt and decrypt specific sets of data.	The management of policies as smart contracts for attribute-based data handling is described in D2.6 [16].
The Wallet should be able to provide access to the device of such VCs and VPs, **only if it is at an expected, correct state**. In order to respond to a request by a device (for example to create a VP or sign a set of traces), it should be verifiable that the Wallet is in a trustworthy state.	The use of key restriction usage policies in the context of the TPM-based Wallet is described in Section 4.1.4.2.
The Wallet must be able to support **enhanced security for allowing secure interaction with the Blockchain**, and must support operations that enable access to attestation-related information to stakeholders **with different levels of granularity**, depending on the level of access that is permitted to be granted to each stakeholder.	The use of the TPM-based Wallet for on-chain interactions is described in D4.4 [1]. Also, the use of VPs in the context of access control is described in Section 4.3.4.
The Wallet must support the **encryption and decryption** of sets of data to be shared based on the presence of a **specific set of attributes**.	The role of the TPM-based Wallet in supporting the ABE scheme is outlined in Chapter 5.
The Wallet must support the **update of the attribute list** of a device to be used in the context of the ABE scheme in case of a secure update.	The key policy update process is described in Section 5.4.4.

Table 1.1: ASSURED TPM-based Wallet and ABE Functional Specifications

1.2 Relation to other WPs and Deliverables

Figure 1.1 depicts the relationship of the deliverable with other Work Packages (WPs) as well as the other tasks in the same WP(4). As aforementioned, the focus of D4.6 is the finalization of the implementation of the TPM-based Wallet the Attribute-Based Encryption (ABE) enabler, which are responsible for the secure storage and management of cryptographic material and various Blockchain interactions, and the attribute-based encryption and decryption of data with varying levels of granularity, respectively. The results presented in this deliverable build upon the initial definition of the TPM-based Wallet presented in D4.5 [13], and provides the final design and implementation of the aforementioned schemes.

The TPM-based Wallet is intended to support the smart contracts related to data sharing and Blockchain data querying, which are presented in D2.6 [16]. In addition, the secure storage and key management capabilities of the TPM-based Wallet support the final version of the ASSURED attestation schemes, such as the Configuration Integrity Verification (CIV) and Control Flow Attestation (CFA) schemes presented in D3.3 [14], as well as the Swarm Attestation scheme presented in D3.7 [15]. The results presented in this deliverable will also provide input to the integration activities of WP5 for the final ASSURED framework, as well as the experimentation activities of WP6 in the context of the ASSURED use cases.

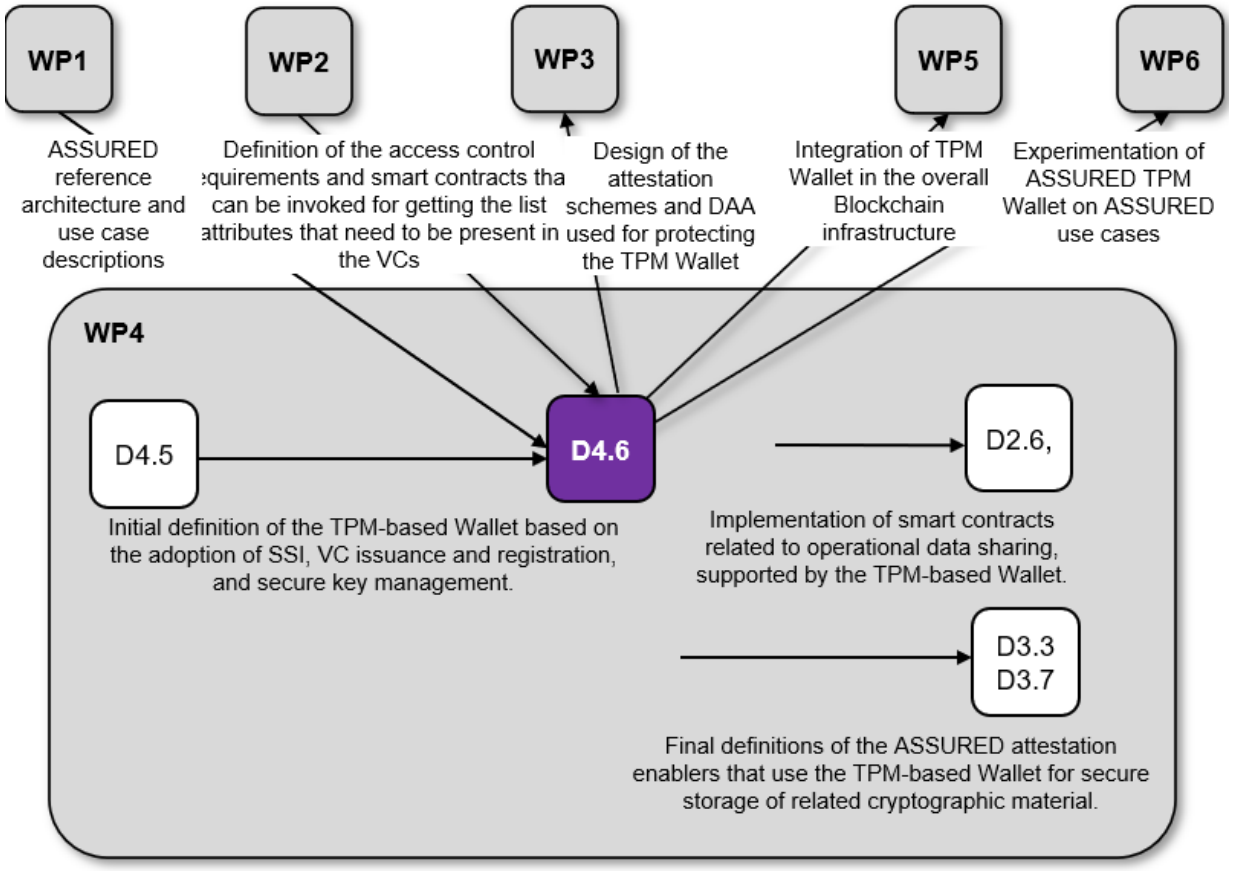

Figure 1.1: Relation of D4.6 with other WPs and Deliverables

1.3 Deliverable Structure

The deliverable is structured as follows: In Chapter 2, we provide a detailed list of the updates and newly implemented features to the TPM-based Wallet and the ABE scheme. In Chapter 3, we define the security and privacy properties that need to be fulfilled by the schemes presented in this deliverable. In Chapter 4, we provide detailed descriptions for all the algorithms and methods related to the TPM-based Wallet, as well as their mathematical security analysis. In Chapter 5, we provide detailed descriptions for the ABE scheme and a mathematical security analysis. In Chapter 6, we provide experimental results for the aforementioned methods. Finally, Chapter 7 concludes the deliverable.

Chapter 2

Summary of Updates in ASSURED TC-based Crypto Enablers for Secure Data Management

The ASSURED TPM-based Wallet offers several features to devices participating in the service-graph chain towards achieving the requirements set forth by the ASSURED use case demonstrators. Specifically, it offers **identity management** through the concept of **Self-Sovereign Identities (SSI)**, which enables a device to make verifiable claims about its identity by leveraging its HW-based Trusted Component (TC). Thus, a device is able to transform its TPM-based Wallet into a trust anchor capable of securely managing **Verifiable Credentials (VCs)**. These are essentially the set of **device attributes** that are issued by the Blockchain CA during the Secure Enrollment process, and include information such as OS version, type of libraries installed, type of CPU microcontroller, as well as security claims on the correct execution of speific software.

The end goal of this approach is to achieve continuous authentication and authorization in the interactions of the devices with the Blockchain infrastructure, either when querying for operational- or attestation-related data, or for recording attestation results (in case of a device acting as a Verifier in an attestation operation). This is achieved by providing devices with the capability to create **Verifiable Presentations (VPs)**, without the need for relying on trusted centralized entities for issuing them. Thus, when a device aims to perform a BC-related operation, it can create a VP based on the aforementioned VCs, containing only the subset of attributes required in order to perform that operation. This approach aligns with the ASSURED goal of operating within a **zero-trust ecosystem**, but also provides scalability by **shifting trust to the devices themselves**, thus eliminating the need for a centralized identity management system.

Based on the above, in ASSURED we have implemented a wide variety of schemes that leverage the TPM-based Wallet of the device and employ VCs and VPs in order to provide attribute-based data management capabilities for data stored on the Blockchain. Specifically, we have implemented the following:

- **Attribute-based Access Control (ABAC)**, which dictates that a set of data stored on the Blockchain can only be accessed by devices exhibiting a specific set of attributes through creating an appropriate VP. ABAC has been implemented and fully documented in D4.5 [13].
- **Attribute-Based Encryption (ABE)**, which enables a device to encrypt a set of data so that only a device that possesses the appropriate attributes is able to perform decryption of

the data by demonstrating the appropriate VP. A first version of ABE has been described in D4.2 [12], and we provide the second version of the scheme in this deliverable.

- **Dynamic Secure Searchable Encryption (DSSE)**, which enables an external entity to perform a search operation on data stored on the Blockchain based on keywords, without decrypting the data themselves. DSSE has been implemented and fully documented in D4.3 [10].

The use of these schemes is based on the aforementioned concept of SSI, which is instantiated in ASSURED with the use of a **TPM-based Wallet**, where we leverage TPMs as the HW-based Trusted Components (TCs), and in this case it is referred to as a **TPM-based Wallet**. The goal of ASSURED in this regard is to provide a **decentralized Blockchain-based identity management system**, which stores different identifiers and attributes using the storage capabilities of the TPM's **Platform Configuration Registers (PCRs)**.

The endmost goal of this approach, and what ASSURED aims to achieve, is to **provide each device full control of its own identity**, and leverages TPMs in order to establish trust across multiple stakeholders and devices when accessing the Blockchain infrastructure for data sharing. Therefore, in the aforementioned attribute-based data management capabilities, the device is able to create VPs as aforementioned, which act as **Decentralized Identifiers (DIDs)** that serve as cryptographically secure, tamper-proof and digitally signed claims, that employ the notion of **selective disclosure** in order to only reveal the attributes required in order to access a particular service or set of data.

First versions of the various mechanisms and schemes implemented with regards to digital identity and data management have been provided in D4.2 [12] and D4.5 [13]. In this deliverable, we provide the second versions of these schemes, which are able to fulfill all the security and privacy requirements set forth by the envisioned use cases. We provide a brief overview of these updates on the ASSURED identity management in Section 2.1, and on the DLT data management schemes in Section 2.2.

2.1 ASSURED Decentralized Digital Identity Scheme

Currently, there is an increasing shift towards decentralized identity models, where there is no centralized governing organization that has control over identity data origination. Specifically, the **World Wide Web Consortium (W3C)** is currently developing two new standards that aim to enable users and devices to produce and manage their own identifiers and credentials without deference or permission from any other administrative organization, namely **Decentralized Identifiers (DIDs)** and **Verifiable Credentials (VCs)**. These are intended to be stored in an Identity Wallet belonging to the device, so that they can be used in order to construct **Verifiable Presentations (VPs)** that serve to prove to a Verifier that it possesses certain attributes, as it was previously mentioned. One core challenge in this approach is the verification of the integrity and origin of the presented VCs or VPs. This translates to the question, *how can someone be sure that these VCs and VPs really belong to the claimed entity?* In addition, *how can a Verifier be sure that the cryptographic key of the device corresponding to a Wallet presenting a VC remains under control of the device, and cannot be used by another unauthorized entity?*

In order to answer these questions, a design choice made in ASSURED with regards to decentralized identity management is the adoption of the notion of **Self-Sovereign Identities (SSI)**. The

goal behind this notion is to provide a user- and device-based Blockchain-based identity management system which aims to store different identifiers and attributes in a digital wallet, which will be under the full control of the users and devices, thus **providing them with the capability to have full control over their own identity, and provide verifiable claims regarding its correctness**. To this end, the first version of the ASSURED TPM-based Wallet was presented in the previous version of this deliverable, D4.5 [13]. Here, we outline the updates on the TPM-based Wallet that will be implemented and integrated into the final version of the ASSURED framework.

One important aspect which was taken into consideration in the construction of the final version of the TPM-based Wallet was the achievement of the required **Level of Assurance (LoA)** behind credential management, depending on the requirements of the target application. The LoA characterizes the degree of confidence in the electronic identification means, thus providing assurance that the person claiming a particular identity is the intended recepient to which the identity is assigned. For example, in Europe, the **eIDAS** regulation clearly defines the requirement for multiple authentication factors to achieve a LoA classified as ”substantial”, and bare proof-of-possession of a SW-based private key does not even achieve the lowest level of LoA in eIDAS. In order to achieve a ”high” level of LoA, the necessary measures are (i) to **isolate the keys from the Holder while still being stored in the user domain**, for example through the incorporation of trusted computing technologies, and (ii) to **bind the identity data to the Holder (Holder Binding)**, based on a unique identifier representing the Holder, such as a secret key. The ASSURED solution aims to *achieve both requirements for higher LoA, while using crypto mechanisms like anonymous credentials to guarantee privacy-enhanced properties*. To the best of our knowledge, ASSURED is the first project of its kind to provide a HW-based Wallet that achieves these properties.

Taking into consideration the above, the final version of the ASSURED TPM-based Wallet introduces a variety of newly implemented features and updates, which brings it in alignment with emerging regulations and standards that require higher LoA for services (such as the aforementioned **eIDAS**). We also ensure that features such as the aforementioned **selective disclosure** are fully supported, towards making the Wallet fully compliant with privacy regulations, such as **GDPR**. To this end, we have designed an **enhanced variant of the DAA-A crypto protocol**, which offers **anonymity**, **unlinkability**, and **unforgeability**, while being the first of its kind to be able to offer guarantees on the integrity of the Wallet when constructing attribute attestations. It also offers the capability to bind identity data to the Holder, by cryptographically binding the Wallet to the intended owner. Therefore, we offer higher levels of confidence to the authentication and electronic identification service of the Wallet, thus a higher LoA as required by emerging regulations such as eIDAS.

In Table 2.1, we provide a list of the updates and features provided by the final version of the ASSURED TPM-based Wallet, as well as their implementation status.

ID #	**Func**	I want to <Action>,	so that <Reason>	**Implementation Status**
WAL_1	**Device Binding and Key Restriction**	As a system operator, I want to be able to bind my VCs to my device using a hardware-protected key	so that it can be guaranteed that only by own TPM can read and interact with the key, ensuring that the cryptographic outcomes are unforgeable.	The use of key restriction usage policies using the ASSURED TPM-based Wallet are presented in Section 4.1.4.2.

WAL_2	**Obtaining a VC**	As a system operator, when I register my device into the ASSURED framework, I want to be able to bind the VC to DAA key	so that my identity and attributes are bound to my device in a hardware-based manner, and I will be able to leverage the bound VCs in order to provide strong verifiable claims regarding my identity and my device properties.	The issuance of VCs during the JOIN phase of the ASSURED TPM-based Wallet protocol is outlined in Section 4.3.2.
WAL_3	**VP Generation and Selective Disclosure**	As a system operator, I want to be able to create a VP as a subset of the issued VCs to use in one of the ASSURED attribute-based cryptographic schemes (such as ABE, ABAC, and DSSE)	so that I can provide only the subset of attributes requested by the target service or Verifier, while simultaneously achieving **full anonymity and unlinkability** of the created VP against both the Privacy CA and the Verifier device, and avoiding threats such as Holder profiling.	The self-issuance of VPs by devices equipped with a TPM-based Wallet is described in Section 4.3.3, and the verification of the VPs is described in Section 4.3.4. In addition, a security analysis of HW-protected VCs for selective disclosure is provided in Section 4.4.
WAL_4	**Restriction to Trusted Configuration**	As a system operator, I want to ensure that my DAA Key is only usable if my device is in a correct operational state	so that, in case my device is compromised by a malicious party, the DAA Key will be rendered inoperable and the attacker will not be able to leverage my Holder device in order to perform various attacks, including impersonation, recreation, and replay attacks.	The restriction of key usage and binding to a trusted configuration is described in Section 4.1.4.2, and is verified through the security proof provided in Section 4.7.
WAL_5	**Key Ownership and Usage**	As a system operator, I want to leverage my TPM-based Wallet to appoint the ownership of my DAA Key to myself	so that a signing operation on a set of data using the DAA Key provides the Verifier with evidence of intention and Holder correctness.	The issuance and binding of the DAA key is performed during the SETUP and JOIN phases of the designed protocol, presented in Section 4.3.1.
WAL_6	**DAA Credential Issuance**	As a system operator, I want to be able to leverage my TPM-based Wallet in order to issue DAA credentials	so that I can participate in operations, such as the attestation enablers provided by ASSURED, in a manner that ensures the achievements of my anonymity and privacy requirements.	The issuance of DAA credentials leveraging the TPM-based Wallet is provided in Section 4.1.4.1 and Section 4.3.1.

Table 2.1: Implementation status of ASSURED DLT data management schemes.

2.2 ASSURED Protocols for DLT Data Management

Towards providing a secure supply chain data management platform for all stakeholders and parties, ASSURED provides a wide array of DLT data management protocols. Therefore, based on the technical **security, privacy, and trustworthiness requirements** defined in D1.1 [9] that need to be achieved by the ASSURED framework when it comes to the management of both **operational** and **threat intelligence (attestation)** data, the developed schemes aim to provide the necessary cryptographic primitives and protocols for safeguarding the secure communication between devices, as well as the secure and privacy-preserving threat intelligence data sharing over the ASSURED Blockchain infrastructure.

In this regard, one of the core innovations of ASSURED is the deployment of a digital **attestation data hub**, as a decentralized infrastructure for certifiable and auditable sharing of both attestation policies and attestation results and their accompanying system traces. Considering the sensitive nature of this type of data, one core consideration of the developed DLT data management protocols was the safeguarding of the security and privacy of this data, in order to prevent a malicious party from obtaining unauthorized access to the attestation evidence, based on which information such as the type of OS running, the type of loaded binaries, and the execution profile of the device could be deduced.

In D4.2 [12], we presented the set of functionalities and algorithms designed in ASSURED, for enhanced data security, integrity, and privacy-preserving mechanisms, which enable **advanced data privacy and ownership safeguarding (privacy by design)**, as well as **data provenance and sovereignty checking mechanisms**. These functionalities, as well as their implementation status, are presented in Table 2.2.

ID #	Func	I want to <Action>,	so that <Reason>	Implementation Status
DLT_1	**Key Management**	As a system operator, I want to ensure that I have access to a key management system that is capable of securely generating, exchanging, storing, protecting, and replacing keys	so that all cryptographic keys are safe and secure during all phases of their storage and use, as a compromised key may lead to massive data leaks, breaches of the privacy requirements, and loss of trust in the system.	The ASSURED key management systems during the enrollment and revocation phases, as well as the Blockchain key management when creating, placing, and querying attestations from the ledger, has been fully described in D4.2 [12].
DLT_2	**Secure Device Enrollment**	If a device wishes to participate in the ASSURED framework, it should get securely onboarded on the overall network, create the required cryptographic material, and perform the issuance of the VCs containing the totality of the usable device attributes	so that the device is able to provide verifiable claims regarding its identity and its operational state, and thus prove that it is who it claims to be and has not been compromised by a malicious party.	The secure enrollment mechanism has been fully implemented and presented in D4.2 [12] and D4.5 [13], and integration with the ASSURED framework has been performed.

DLT_3	**Property Based CFA and CIV**	As a Prover device, I want to be able to generate static, boot-time or load-time evidence of my system component's correct configuration (CIV), as well as guarantees on the correctness of control- and information-flow properties (CFA)	so that I can prove to a Verifier device issuing an attestation challenge that I am in a correct operational state, based on the correctness of the requested device attributes.	The ASSURED attestation mechanisms have been analyzed in detail in D3.2 [11] and D3.3 [14]. Integration with the ASSURED framework has been performed, and the updated versions of the attestation schemes will be completed during the second reporting period.
DLT_4	**Pseudonyms**	As a Prover device, I want to be able to sign data using a short-term anonymous key pair (pseudonym)	so that I can participate in attestation processes while retaining my anonymity, without disclosing information regarding my identity to the Verifier.	The use of pseudonyms in ASSURED has been described in D4.2 [12], and integration with the ASSURED framework has been performed.
DLT_5	**Revocation**	As a system operator, I want to be able to revoke a device based on its pseudonym following a failed attestation	so that I can remove a potentially compromised or malicious device from the network, without violating the anonymity of the device and the privacy requirements of the system.	The revocation scheme implemented in ASSURED has been described in D4.2 [12], and integration with the ASSURED framework has been performed.

Table 2.2: Implementation status of ASSURED DLT data management schemes.

One of the core functionalities of ASSURED with regards to secure data management is **Attribute-Based Encryption (ABE)**. ABE is essentially an expressive way to define asymmetric-key encryption schemes for policy enforcement based on attributes, where both a user secret key and a ciphertext are associated with sets of attributes. In the context of ASSURED, as aforementioned, these attributes are defined in the form of VCs, which are issued by the Privacy CA during the secure enrollment of the device, and based on which an Encryptor device can select the appropriate subset of attributes to perform an encryption process based on the corresponding policy. Afterwards, a Decryptor device can create a VP containing the required attributes based on its VCs, in order to decrypt this data.

In the context of ASSURED, ABE is an important tool towards achieving the security and privacy requirements set forth by the envisioned use cases. For example, consider the *DAEM Smart Cities* use case, which involves the handling of various kinds of operational data, which may originate from devices such as smoke sensors or surveillance cameras. In this case, we need to provide access to such encrypted data with the required level of granularity in order to capture the different kinds of stakeholders. For example, it is possible an encrypted set of data should be able to be decrypted only by a specific kind of user, such as firefighers or police officers. It follows that there are strict security and privacy requirements which need to be fulfilled by the ASSURED ABE scheme, which are outlined in Section 5.2.2.

As aforementioned, the first version of the ASSURED ABE scheme was presented in D4.2 [12]. In this deliverable, we will present the updated version of the scheme, which is able to capture

these requirements. Specifically, the features of the ASSURED ABE, including those newly implemented in the second version, are given in Table 2.3.

ID #	Func	I want to <Action>,	so that <Reason>	Implementation Status
ABE_1	**Attribute-based encryption/decryption**	As a system operator, I want to be able to encrypt a set of data by using a key generated based on a specific device attributes, as dictated by an appropriate security policy	so that the only devices able to decrypt the data are those who are able to verifiably exhibit the specified attributes through a VP.	The first version of the ABE scheme, presented in D4.2 [12], is able to perform attribute-based encryption and decryption of sets of data.
ABE_2	**Attribute list correctness verification**	As a system operator, I want to ensure that a set of data encrypted with ABE can only be decrypted with the same policy employed during encryption	so that the data cannot be decrypted by a malicious party who attempts to bypass the policy by using an outdated policy (i.e., one that became deprecated following a secure update process).	In the second version of ABE, we include a hash of the policy itself in the seed used to generate the encryption key, thus guaranteeing that the Encryptor and the Decryptor use the same policy. This is further outlined in Section 5.4.
ABE_3	**Privacy-preserving encryption**	As a system operator, I want to ensure that during an ABE encryption process, I do not divulge any personally identifiable information, or information that can be used to deduce implementation details	so that a malicious party cannot use my produced signatures in order to perform implementation disclosure attacks.	In the second version of the ABE scheme, no device is linked with the encrypted data, since the employed keys are either **ephemeral** (Signing Key, ECC Key), or **are not publicated** (Encryption Key, HMAC). We expand on this approach in Section 5.4.
ABE_4	**Internal integrity check of Encryptor**	As a system operator, when I encrypt a set of data using ABE, I want to ensure that the operational state of my device is correct	so that, in case my Host Device has been compromised by a malicious party, I will not be able to encrypt data using ABE and thus prevent any further actions that the attacker aims to take.	Prior to the encryption of data, the Encryptor device performs (i) a **PolicyPCR** command to load the attribute keys from the PCRs and (ii) a **PolicyOR** with the golden hashes of the runtime attributes from the Tracer in order to perform an integrity check. If this check is unsuccessful, encryption cannot be performed. This is further analyzed in Section 5.4.

ABE_5	**Internal integrity check of Decryptor**	As a system operator, prior to decrypting a set of data which has been encrypted using ABE, I want to ensure that my Host Device has not been compromised by a malicious party	so that I can ensure that an attacker that may have hijacked my device cannot obtain unauthorized access to the encrypted data.	Similarly to **ABE_4**, prior to the decryption of a set of ABE-encrypted data, the Decryptor (i) executes a **PolicyPCR** command to ensure that it possesses the correct attributes and (ii) a **PolicyOR** to verify that it is in a correct operational state. If this integrity check is unsuccessful, decryption cannot be performed. This is further analyzed in Section 5.4.

Table 2.3: Features of ASSURED ABE scheme.

Chapter 3

System and Trust Model for ASSURED Secure Data & Identity Management

Here, we define the **security requirements and properties** that need to be considered in the implementation of the TPM-based Wallet and the ABE scheme, in order to achieve the envisioned design approach with regards to identity and data management which was outlined in Chapter 1. Note that some of the properties that will be presented are overlapping between the two schemes. For example, the **unforgeability** property dictates that it should not be possible for a potentially malicious third party to produce forged signatures that appear as if they originate from the original user or device. This is applicable to both schemes, but it takes a different meaning for each. Specifically, for the TPM-based Wallet it refers to forging credentials, while for the ABE scheme it refers to forging encrypted data. For the sake of **completeness** and for providing **very well defined properties based on which the security analysis of each scheme will be performed**, we have provided separate property lists for each scheme.

3.1 Security Properties Definition

As it was previously mentioned, this deliverable is dedicated to a detailed presentation of the final versions of the ASSURED TPM-based Wallet and the ASSURED Attribute-based Encryption (ABE) scheme. Here, we define the set of **security and requirements** that have to be fulfilled by these components. To this end, in Tables 3.1 and 3.2 we provide the requirements defined for the TPM-based Wallet and the ABE scheme, respectively. Note that, in Chapter 6, we will provide a detailed formal security analysis of the TPM-based Wallet and the ABE scheme, where we will demonstrate the fulfillment of these security and privacy requirements.

ID	Requirement	Description
WAL_SR1	**DAA-VC Credential Device Binding**	If at least one of the issuers is honest, it should only be possible for a Verifiable Credential (VC) to be issued by the Privacy CA, to the TPM-based Wallet that has a unique TPM key tsk. Moreover, the issued VC should be binded to a unique TPM key, meaning that no adversary should be able to create a Verifiable Presentation (VP) using a TPM key that is not binded to its VC. At the end of the secure enrollment process, the tsk of the TPM should be binded to the corresponding approved attribute set $attr$ issued for the Wallet. Whenever the Wallet requires a signature, a check is performed in order to verify that it has been correctly registered.

WAL_SR2	**Holder Binding**	It must be ensured that identity data issued by the Privacy CA is only delivered to the intended holder. This serves to safeguard the Wallet against adversaries who aim to present valid attribute attestations in the form of a VP, without having access or being the intended recepient of the issued credential, certifying that these claims are bound to this specific Holder. For example, an adversary may obtain access to a Holder's VC, and attempt to construct VPs that will be accepted by a Verifier. In this context, there should be appropriate mechanisms to ensure that the Holder is tightly bound to the credentials at the time of the presentation of VP, while simultaneously protecting the Holder's privacy.
WAL_SR3	**Correctness of Wallet State**	A TPM-based Wallet should only be allowed to use a DAA Key to sign a VC or create a VP if no compromise has been detected. Therefore, a Verifier will accept a presented claim, if and only if the Wallet can provide verifiable evidence that its integrity has not been compromised (from the time of credential issuance) in an unauthenticated manner. This essentially dictates the enforcement of key restriction usage policies for governing the credential management, leveraging the policy-based safeguards of the TPM.
WAL_SR4	**Full Anonymity**	An adversary that is given two signatures should not be able to distinguish distinguish whether both signatures were created by one honest TPM-based Wallet or not. Also, no adversary should be able to link signatures on two messages m_1 and m_2, even if these signatures were created by the same honest signer. Full anonymity of signatures created by an honest TPM $\mathcal{M}$ together with an honest Trusted Component (TC) bridge is guaranteed with the ASSURED TPM-based Wallet due to the random choice of tsk for every signature, which guarantees the anonymity of the Wallet. Furthermore, our implementations creates signatures on behalf of honest Wallets, where the undisclosed attributes are set to dummy values. This guarantees that the signature leaks no information (perfectly hides) the committed attributes, but only reveals the number of total attributes, thus achieving full anonymity.
WAL_SR5	**Key Uniqueness**	It should be ensured that any newly registered DAA Key is not being used by any existing device. This is achieved by a check performed during the Secure Enrollment phase.
WAL_SR6	**Correctness**	If a signer device creating a signature on a set of data and the Verifier are honest, the signatures generated by the signer should be accepted by the Verifier with overwhelming probability. In addition, each signature should always be traced to the signer who created it, and the provided signature should not match any revoked key to ensure a correct revocation. The Privacy CA is in charge of the attributes and must explicitly allow a TPM-based Wallet to be issued certain attributes in the Secure Enrollment phase. In addition, during the creation of a signature, it should be verified whether the signer has the correct attributes, and it should not be possible for a TPM-based Wallet to create valid signatures with respect to attribute predicates that do not hold for the attributes of this Wallet.
WAL_SR7	**Unforgeability**	No adversary should be able to create a signature σ_D on a message m on behalf of a TC bridge, when this specific TC Bridge (with the underlying TPM) has not signed m. The signature should be able to be traced to one TPM tsk, and if the TPM is honest, any signature on messages not signed by that TPM should be rejected. We highlight that our Unforgeability notion is stronger than the notion considered in various VC schemes with only one issuer, where the DAA Issuer can forge credentials and thus create forged signatures. Our unforgeability definition doesn't allow the DAA Issuer alone to forge signatures nor the Privacy CA alone, unless they are both corrupt, which provides stronger unforgeability guarantees.

Table 3.1: Security & Privacy Requirements for ASSURED Identity Management

ID	Requirement	Description
ABE_SR1	**Unfogreability**	It should not be possible for a malicious entity to create encrypted messages based on attributes it does not possess. Consider, for instance, a device with the ASSURED **Trusted Computing Base (TCB)**, but whose host has been compromised. Also, consider the existence of a policy, which dictates which static and dynamic attributes should be used by the device in order to encrypt a set of data. In this case, it should not be possible for the compromised device, even it has been securely enrolled in the system, to either bypass the policy with the list of attributes, or to use a deprecated policy, in order to create local encryption keys. Thus, it should not be possible for a compromised device to forge encrypted messages, or to create messages under an illegitimate policy. To achieve unforgeability, we assume that the device's TCB is able to manage key generation, after validating that the correct attributes are present. Therefore, the creation of the keys takes place in the TPM, so that a malicious party cannot create keys and forge encrypted messages if the correct attributes are not present.
ABE_SR2	**Non-frameability**	It should not be possible for a malicious party to encrypt a set of data in a manner that appears as if the encryption operation has been performed by a different party. This is needed so that the ABE scheme can prevent **Sybil attacks**, which rely on a compromised device impersonating another device, which aims to perform an encryption operation on a set of data, while claiming to be the other device. Specifically, Sybil attacks entail the creation of **ghost nodes** that aim to frame or impersonate legitimate nodes, and generate encrypted data that appears as if it originates from their side. This type of attacks may be used in order to compromise the identity of the legitimate devices. This is achieved through the signing of the attribute keys with the **randomized credential** of the device, which ensures that anyone can understand that the encrypted message originates from a valid TPM, but is not able to deduce the identity of the device. Non-frameability is also ensured during the **zero-touch onboarding** process of the device with the help of the SCB, which acts as the trusted entity in the ASSURED ABE scheme and is considered to be the attribute holder. During the onboarding process, the attributes of the device are securely registered in the form of **Verifiable Credentials (VCs)**.
ABE_SR3	**Provenance**	Provenance can be expressed in two different aspects. (i) When data is encrypted by a device using ABE, it should be possible to verify that the encryption has been performed by the correct device, which possesses the appropriate set of attributes. In other words, an entity with the correct privileges should be able to **verify the identity of the device that encrypted the data**, by checking both that **it has access to the correct attributes**, and that **it is the appropriate owner that verified the particular data**. (ii) A device that encrypts data with ABE should be able to do so while ensuring its identity privacy. However, it should also be possible for an entity with the appropriate privileges to be able to verify the provenance of the data that originates from the intended data owner device that has the correct attributes. This is achieved with the use of **randomized DAA credentials** for the signing operation, which enable a party with the required privileges to communicate with the SCB in order to link the credential with the identity, through the controlled linkability feature of the enhanced ASSURED DAA scheme D3.2 [11].

ABE_SR4	**Proof of attribute list correctness**	It should be provable that, during an encryption process, the corresponding policy used is correct. This means that it should be provable that the policy circulated and enforced by the SCB (i) should not be deprecated (i.e., replaced by a newer policy following a secure update of a binary), and (ii) specifies the correct set of attributes. In order to achieve this, as it will be shown in Section 5.3, every time the Encryptor device receives a policy containing an attribute list based on which ABE should be performed, the **hash of the policy** is contained in the seed that creates the HMAC and the local Encryption Key in the Encryptor device, along with a **nonce** and the **keyed hash** taken from the SCB. Then, if the Decryptor has access to the same the same keyed hash and attribute list, it creates the appropriate decryption keys. If, for instance, the Decryptor has the same list of attributes but not the same policy (i.e., if the policy used by the Encryptor was compromised), the **Key Derivation Function (KDF)** will not provide the Decryption Key that corresponds with the Encryption Key used by the Encryptor, so the KDF function will fail. Thus, we can implicitly capture any deviation in the policies used by the Encryptor. In addition, after the encrypted data is recorded on the ledger, it should be possible for **any party with the required privileges to verify that the encryption policy was correct**. This is achieved by creating a certificate containing the policy loaded at the time the keys were created, which is afterwards recorded at the ledger, accompanying the encrypted data.
ABE_SR5	**Secure attribute policy update**	It should be possible to update an attribute policy, in case a change in the corresponding attribute list is required (e.g., in the case of a secure update). Consider, for instance, that the SCB creates a policy that dictates a specific set of attributes that is needed in order to encrypt and decrypt a set of data. During the operational lifecycle of the system, it is possible that **the policy may need to be updated** (e.g., in case a new attribute has been added to a particular binary). In this case, the SCB should be able to appropriately update this policy. But in this case, it is also important to consider that this should be done in a way that makes it **impossible for the device to possess two active policies simultaneously**, i.e., the previous policy and the new updated policy. This ties into **ABE_SR4**, which ensures that the deprecated policy will not be used for an encryption operation. This is because a compromised host device may aim to keep the previous policy, so that any time an encryption process needs to be performed, it will be able to **load the old policy in its TPM and bypass the new policy**, which is prevented as previously outlined.

Table 3.2: Security & Trust Requirements for ASSURED Attribute-based Encryption.

Chapter 4

ASSURED TPM-based Wallet

Adopted from the traditional SSI framework, a holder requests a **Verifiable Credential (VC) consisting of a number of claims from a credential-issuing authority**. For example, a holder might request a certificate from a government agency, containing attributes (claims) of the holder's DOB, name, address, and that the holder is over the age of 18. The holder might be requested to prove that he is above the age of 18, however, the holder does not wish to disclose his real birthday. The acquired credential can then be crafted into a **Verifiable Presentation (VP)**, which only discloses a subset of the claims necessary, i.e., the claim that the holder is over the age of 18. A Verifier can verify that the claims in the VP are cryptographically correct and were issued by a trusted issuer, validating the holder's age.

In the traditional approach, the Verifier cannot verify that a credential was not transferred between holders, nor can it confirm that the holder intended to craft this presentation (whether due to malware or otherwise). Furthermore, there are no documented guarantees that presentations cannot be linked together, which enables a Verifier (or colluding Verifiers) to acquire more and more knowledge about the holder over time.

To provide these securities, we present an updated architecture in which a **Trusted Component (TC) is incorporated in the Holder's device and a secondary issuer is available to the Wallet**. This issuer, called the **DAA Issuer**, represents an authority that possesses the capabilities to validate the Trusted Component (TC). This can be, but is not limited to, the manufacturer of the component. This issuer is used in association with the TC (and possibly the VC Issuer) to generate a DAA key that guarantees the **correctness of the holder**, ensures that **the credentials are non-transferable**, and proves that **presentations are made with an authenticated intent**. Moreover, such a key, using the DAA-scheme provides **user-controlled unlinkability** and anonymity between signatures, hindering a Verifier from obtaining knowledge over time.

4.1 ASSURED and Decentralized Digital Identity Wallets

4.1.1 System Model

For the construction of the ASSURED TPM-based Wallet, we utilize an **enhanced version of the DAA scheme** which was presented in D3.2 [11]. Recall that DAA is an anonymous signature scheme, which allows a TC to attest to the state of the host system, while preserving the privacy of the Holder. A DAA scheme consists of an **Issuer** (i.e., the **DAA Issuer**) and a set of **signers** (i.e., the **Holders**). It includes five algorithms: **Setup**, **Join**, **Sign**, **Verify**, and **Link**. The DAA Issuer produces a DAA membership credential for each Holder, corresponding to a signature on

the unique identifier of the Holder. This is the **hardware-based DAA Key**, which is stored inside the TPM, and its usage is safeguarded through a number of policy regulations, governed by the **TC Bridge**, which acts as the mediator between the Wallet and the underlying TPM. This enables the secure and authenticated use of the DAA Key, as well as the verification of the Holder and Device Binding properties. DAA provides two signature modes: (i) **fully anonymous signatures**, and (ii) **pseudonymous signatures** that provide user-controlled linkability.

In the updated version of DAA, referred to as **Attribute-based DAA (DAA-A)**, the difference is that the public key does not correspond to a singe secret key, but is the result of a **discrete logarithmic representation of multiple attributes**. This property enables us to create VPs fulfilling the **selective disclosure** property by encoding each attribute as a separate key, while also providing complete anonymity by allowing the representation of identity as a separate attribute key to be hidden, thus fulfilling the **zero-knowledge** design principle followed in ASSURED.

Also, note that the notion of the **unforgeability** property (outlined in Table 3.1) is stronger than what is required in existing VC management schemes, where only one Issuer is assumed in the system model, where the Issuer may aim to forge credentials in order to create forged signatures on presented claims. In ASSURED, we adopt the **separation-of-duties** principle, based on which we split the issuance entities into the **DAA Issuer** (for Device Binding and Key Restriction operations) and the **VC Issuer** (for VC Issuance operations). This ensures that no single Issuer entity can forge signatures.

4.1.2 Threat Model

The challenge that all key-management systems must answer is how to protect against the compromise of a private key via any of the myriad ways that can happen. This may include (i) **lost or stolen devices**, (ii) **security flaws in the digital Wallet**, (iii) **side-channel attacks on the digital Wallet** (key leakage), (iv) **social engineering attacks on the controller**, and (v) **Extortion attack on the controller** (rubber-hose cryptanalysis). Private key compromise is even more dangerous in decentralized key management because there is no authority higher than the controller of the keys. So losing control of a private key is equivalent in seriousness to handing over control of all DIDs that depend on the private key.

We can capture any attack that leaves a footprint on the device state, such as **code injection attacks**, **buffer overflows**, **memory-related vulnerabilities**, etc. More advanced attacks that can compromise a device without leaving a footprint, such as **return-oriented programming (ROP)** and **control flow hi-jacking**, cannot be directly captured. However, these attacks can also be detected if the Issuer can send what should be the expected execution behavior of a device to protect the usage of the key.

4.1.3 Assumptions

We assume that the software stack, as well as its interactions with the underlying root of trust (TPM), are trusted. Therefore, the TPM cannot be used as an oracle and be leveraged by an attacker for issuing valid responses.

In addition, we assume that the attacker cannot impersonate the Holder by getting the proof of knowledge to access and interact with the Wallet.

4.1.4 Preliminaries

4.1.4.1 Attribute-based Direct Anonymous Attestation (DAA)

As it was previously mentioned, **Direct Anonymous Attestation (DAA)** [5] is an anonymous signature scheme, which allows the Trusted Platform Module (TPM) to attest to the state of the host system while preserving the privacy of the user. From a high-level view, the DAA-A construction [7] consists of an underlying DAA scheme with an anonymous credential of a public key. But in contrast to a DAA scheme, the public key does not correspond to a single secret key but is the result of a discrete logarithm representation of multiple attributes.

The DAA-A protocol doesn't only provide privacy of the user's identity that is achieved via the DAA scheme, but also offers extra **privacy for non-disclosed user attributes**. For each undisclosed attribute x_k, a signature of knowledge on x_k in the form of $s_k = \omega_k - cx_k$, for some random ω and a challenge c, is created which leaks no information about the attribute x_k due to the zero-knowledge property of Schnorr signature. Relying on these strong privacy guarantees, the ASSURED TPM-based Wallet is constructed upon the DAA-A scheme, by adding extra layers of security. In order to achieve this, we construct **policy regulations to govern the usage of the DAA Key** by the Holder (key restriction usage policies) to sign attribute claims. This can ensure the binding of the issued identity data to the Holder, and is performed by the Privacy CA through binding the issued attributes to the anonymized part of the DAA credential. In addition, we construct a policy to restrict the usage of the DAA key to creating attribute attestations, **if and only if the integrity of the Wallet has not been altered in an unauthenticated manner**.

Although the DAA-A protocol has been primarily designed for use with a TPM chip, it can also be used in other contexts, for instance as an **identity smart card**. In this case, the complete signing operation of the protocol has to be performed on the smart card. The **CL-based DAA-A protocol** is especially well suited for this because it uses only standard elliptic curve calculations.

4.1.4.2 Key Restriction Usage Policies

The TPM consists of four hierarchies: **Endorsement Hierarchy (EH)**, **Platform Hierarchy (PH)**, **Owner Hierarchy (OH)**, and `NULL` **Hierarchy (NH)**. The EH is primarily used for privacy-sensitive data and PH for low-level operations, e.g., in conjunction with firmware. We will be using OH as it is used for user operations, as is the `NULL` hierarchy. A TPM hierarchy contains a cryptographic seed that forms a foundation for key generation.

Using the **seed and input parameters (key template)**, the TPM can generate a *Primary Key*, a key that de facto lies in the volatile memory but can be made persistent. As long as the template and seed remain the same, the key is recreatable ad hoc - an essential property due to the TPM's limited memory size.

The **TPM can create a new key as a child of a primary key**, and these will always be unique. The parent key will encrypt such a key's private area and eject it onto the host, meaning it is unusable outside the TPM (a public part of the key is still functional, e.g., to verify signatures). The primary key must be present in the TPM memory to decrypt the key's private part, making it functional. **The private area of a key can contain a password** for use, and **the public area can contain a policy**, which we will further analyze in the following section. The hash of the public area is the key's fingerprint and is called the **name of the key**.

4.1.4.3 Policies and Sessions

The TPM 2.0 specification [22] allows one to define a policy and **require specific assertions or actions to take place before access to a protected object is allowed**. The policy associated with an object is represented internally with a single statistically unique digest value known as the **policy digest**. Access to all objects making use of this enhanced authorization takes place via a session-based authorization procedure, during which the caller issues a sequence of policy commands to the TPM. Each policy command is an assertion that a particular statement is true in order for the policy to be satisfied. In this case, the policy command modifies a digest value associated with the session, characteristic of the particular policy expressed via the sequence of policy commands. This running accumulation of the digest value is called the **session digest**.

Multiple policy commands will form a unique session digest based on the policy parameters and the TPM's internal checks. After the policy command sequence has been completed, the final value of the session digest is compared to the policy digest of the object being accessed. A match indicates that the sequence of invoked policy commands satisfies the assertions expressed by the policy, and authorization is granted.

4.1.4.4 Non-volatile Indexes

The TPM offers the capability to create persistent indexes of variable size within the module's protected domain, with different properties. Highlighted here are the ”bit index” and ”PIN pass index” types, both with a size of 8 bytes. The bit index can only be written to using an exclusive OR operation of another 8 bytes. The latter includes an authorization value, or PIN, and contains a counter for the current number of valid authorizations as well as an authorization limit.

When an object with a policy, e.g., a key, is used in operations, **the TPM verifies that the session digest matches the policy digest**. So, before the key is functioning, all policy commands must be executed (in the correct order) to manipulate the session digest. Suppose that a single command does not provide the correct hash (for example, `PolicyPCR` embeds a PCR value into the session digest). In that case, the resulting hash will not match the key's policy digest, and the TPM will reject the operation. The policy digest of an entity is public and can be manipulated by the user after creation; however, due to integrity checks with references in the private parts, this will be discovered by the TPM, and it will reject the operation.

4.2 Technical Description of the ASSURED TPM-based Wallet

In Figure 4.1, we provide a high level overview of the functionalities provided by the ASSURED TPM-based Wallet, including the conceptual flows between the actors participating in these functionalities. In the Figure, steps 1, 2, 3, and 4 pertain to **Credential Management**, while steps 5 and 6 cover **Attribute Authentication**. Also note that, in the context of ASSURED, the **Privacy CA** acts as the **VC Issuer**, while the **DAA Issuer** can either be a different entity that is responsible for certifying the DAA Key, or can be the Privacy CA itself. These are the two governing entities that govern the **Secure Enrollment** process in ASSURED in order to be able to certify the correct creation of the DAA Key, binded to policies that represent the correct state of the device.

The core functionalities of the ASSURED TPM-based Wallet that are depicted in Figure 4.1 are as follows:

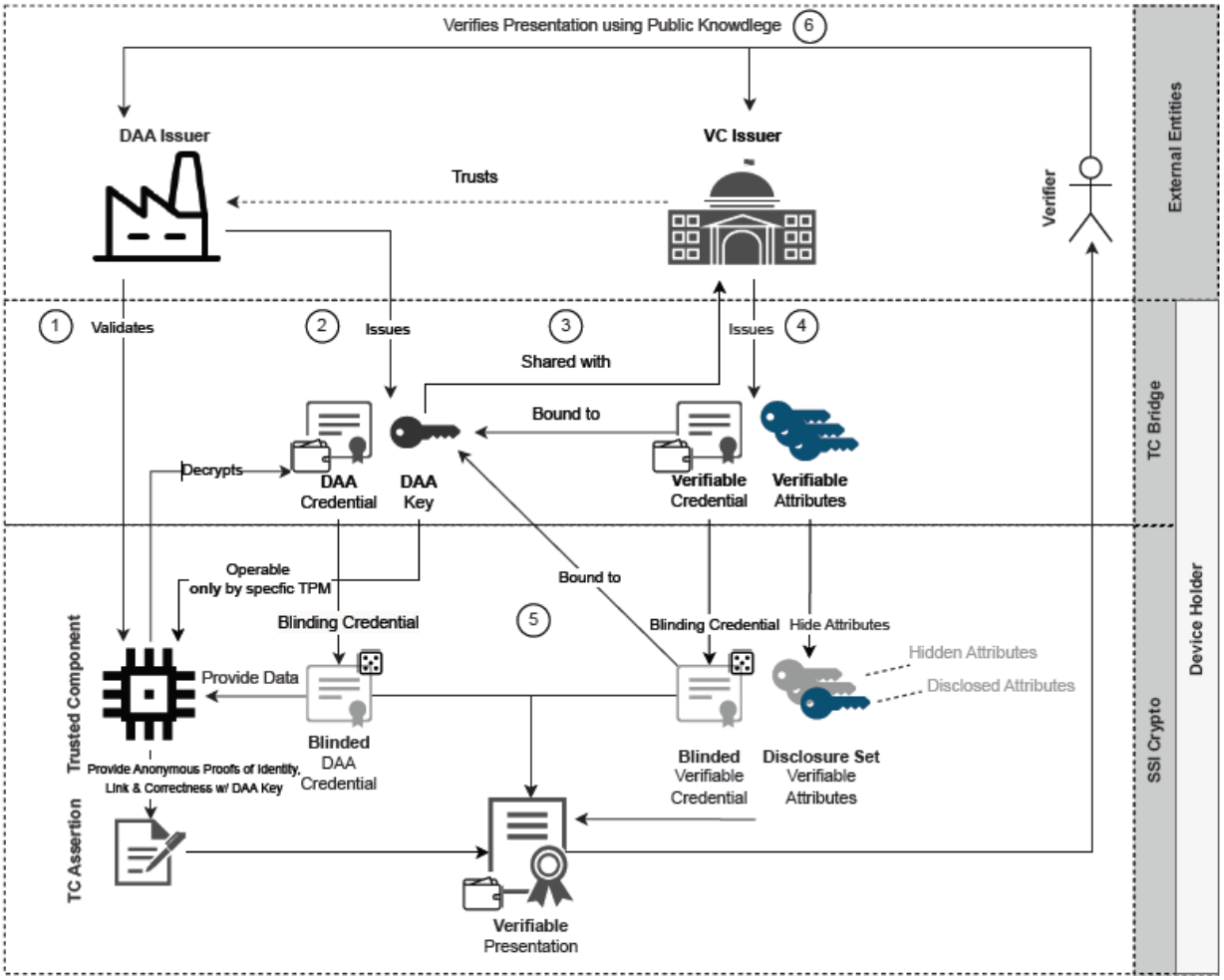

Figure 4.1: TPM-based Wallet High-level Architectural Overview and Credential Management Functionality

- ***Device Binding and Key Registration:*** To support the Device Binding of VCs, we use a hardware-protected key, built into the Holder device's TPM. This guarantees that only this particular component can read and interact with the key, thus ensuring the unforgeability of the cryptographic outcomes. Before the creation of the key, the DAA Issuer and Privacy CA can negotiate the requirements dictating when the key can be used. Upon agreement of the requirements, the DAA Issuer creates the corresponding **key restriction usage policy** and sends it to the TC Bridge. Then, the TC builds the key and releases its public part, as well as integrity-protected information (such as the aforementioned policy). This information is shared with the DAA Issuer to verify the key and validate the TPM (**Step 1**). If both checks succeed, a **DAA Credential** for the key is released (**Step 2**). This process, which is part of the **Secure Enrollment** process in ASSURED, is presented in Section 4.3.1.
- ***Obtaining a Verifiable Credential (VC):*** When requesting a VC, the TC Bridge shares the DAA public key with the Privacy CA (**Step 3**) and authenticates the Holder device, along with a DAA signature to prove device ownership. Then, the Privacy CA constructs the relevant attributes to be included in the VC, and includes the DAA public key as an **identity attribute**. This process **binds the VC to the DAA Key**, thus binding the VC to the Wallet. The credential is returned to the Holder and stored in the Wallet (**Step 4**).
- ***Generating a Verifiable Presentation (VP):*** The Wallet can create a VP consisting of a

subset of the attributes contained in the VC, which can be verified by any Verifier who knows the VC and the corresponding DAA credentials. a **"binding" or "randomization"** operation (**Step 5**) is implemented in order to protect against VP linkability (which may lead to Holder profiling) and ensures **Full Anonymity** and **Unlinkability**, while not negating the aforementioned Device Binding property that verifies the device identity. This process also fulfills the **selective disclosure** property, by enabling the device to choose which attributes to disclose, while hiding all undisclosed attributes with a cryptographic operation that attests to their equivalence in the original VC, thus providing the necessary proof that the VP has been correctly created by the Holder. Note that the DAA Private Key (i.e., the identity attribute) is also hidden in the TPM in order to preserve anonymity. The final VP blob s constructed as a set of disclosed and undisclosed attributes accompanied by the two binded credentials and can be shared with the Verifier (**Step 6**). Then, the Verifier can use the mechanisms provided by the DAA-A scheme in order to **verify the validity of the attributes**, to **confirm that the Holder controls the undisclosed attributes**, and the **correct key restriction usage policy has been implemented**. The VP creation process is described in detail in Section 4.3.3, and the VP verification process is described in Section 4.3.4.

4.2.1 Notation

Next, we provide some notations that will be used in the technical descriptions of the aforementioned functionalities. Let $\mathbb{F}$ be a **finite base field** and $\tilde{\mathbb{F}}$ be a **finite extension field** of $\mathbb{F}$. Let $\mathbb{E}$ be an **elliptic curve** defined over $\mathbb{F}$ with a base point G_0. Let $\tilde{\mathbb{E}}$ denote the points of $\mathbb{E}$ over the extension field $\tilde{\mathbb{F}}$ and $\tilde{G_0}$ be a base point of $\tilde{\mathbb{E}}$. The curve $\mathbb{E}$ shall be equipped with a **type III pairing** $\tau : \mathbb{E} \times \tilde{\mathbb{E}} \to \tilde{\mathbb{F}}$. Uppercase Latin or Greek letters always indicate elliptic curve points on the curve $\mathbb{E}$. Uppercase Latin or Greek letters with a tilde on top will denote elements on the curve $\tilde{\mathbb{E}}$. The operation on $\mathbb{E}$ (resp. $\tilde{\mathbb{E}}$) is written with additive notation. Multiplication by scalars is always written on left. Scalars are always defined on $\mathbb{Z}_q$, where q is the group order of the subgroup $\langle G_0 \rangle$ in $\mathbb{E}$. Arithmetic has to be understood in the respective finite fields. The hash function: $\mathsf{H}_1 : \{0,1\}^* \to \mathbb{Z}_q$ is used in this scheme.

4.2.2 High Level Protocol

Our protocol involves five types of players: a DAA-issuer $\mathsf{I}_{\mathsf{DAA}} \in \mathbb{I}_{\mathbb{DAA}}$, a VC issuer $\mathsf{I}_{\mathsf{VC}} \in \mathbb{I}_{\mathbb{VC}}$, a set of TPMs and their corresponding hosts, and a Verifier $\mathcal{V} \in \mathbb{V}$. For simplicity of our scheme description and model, we assume the existence of one DAA issuer and one VC issuer, and we denote a TPM by $\mathcal{M} \in \mathbb{M}$ and its host by $\mathcal{H} \in H\mathsf{OST}$.

Our protocol consists of the following four polynomial-time algorithms (Setup, Join, Sign, Verify):

Setup:

- DAA-Issuer Setup: On input of a security parameter 1^α, the DAA-issuer uses this randomized algorithm to produce a pair of values (isk_{DAA}, ipk_{DAA}) as the DAA issuer's secret and public key pair.
- VC-Isssuer Setup: On input of a security parameter 1^β, the VC-issuer uses this randomized algorithm to produce a pair of values (isk_{VC}, ipk_{VC}) as the VC issuer's secret key and public key pair.

The global public parameters for the system include both issuers' public keys and are denoted by $pp = (ipk_{DAA}, ipk_{VC})$.

Join: This protocol is performed between a signer that consists of a TPM $\mathcal{M}$ and a host $\mathcal{H}$, the DAA issuer I_{DAA} and the VC issuer I_{VC}. $\mathcal{M}$ inputs a secret DAA key tsk and its associated public DAA key PK. I_{DAA} produces cre_{PK} that is a DAA credential associated with (tsk, PK). I_{VC} selects a set of attributes $\{att_i\}$ and their corresponding keys $\{tsk_i\}_{i\in[1,n]}$, and produces a credential cre_w on $\Gamma = PK + \sum_{i=1}^{k=n} tsk_i G_i$. As a result, $\mathcal{M}$ saves tsk, while $\mathcal{H}$ holds $\{att_i, tsk_i\}$ for each attribute and (cre_{PK}, cre_w).

Sign: On input of tsk, $\{att_i, tsk_i\}_{i\in[1,n]}$ (with an indication of hidden/open attributes), cre_{PK} and cre_w, a message m that includes the data to be signed and the Verifier's nonce n_V for freshness, $\mathcal{M}$ and $\mathcal{H}$ run this protocol to produce a randomized signature σ on m under (tsk, cre_w, cre_{PK}).

Verify: On an input consisting of m, a candidate signature σ for m, and a set of rogue signers' secret keys RogueList KRL, the Verifier $\mathcal{V}$ uses this deterministic algorithm to return either 1 (accept) or 0 (reject).

4.3 Architectural Details

The protocol is based on [7] with two changes to meet our application requirements: (I) supporting **unlinkability**; (II) introducing the **VC issuer** I_{VC} in addition to I_{DAA}, where I_{DAA} only certifies the TPM public key and I_{VC} certifies the Wallet attribute keys.

Note that in this section, we provide the technical descriptions of the actions and cryptographic operations performed in the context of each of the phases corresponding to methods performed by the TPM-based Wallet. In Section 7.1, we provide descriptions of the implementation of these phases, considering also TPM command execution.

4.3.1 SETUP & JOIN Phases

Setup:
The public group elements $G, G_0, G_1, \ldots, G_n \in \mathbb{E}$ and $\tilde{G}, \tilde{G}_0, \tilde{G}_1, \ldots, \tilde{G}_n \in \tilde{\mathbb{E}}$ corresponding to the TPM's key and token attributes, where $G = r_G G_0$ and $G_k = r_k G_0$, $\tilde{G} = r_G \tilde{G}_0$ and $\tilde{G}_k = r_k \tilde{G}_0$ for $k = 1 \ldots n$ and $r_G, r_k \in_R \mathbb{Z}_q$. It is required that the values of r_G and r_k for $k = 1 \ldots n$ are generated by the setup system and erased after the setup process, such that there is no known discrete logarithm relation between any G_k and G_j and between any G_k and G.

A signer generates its authentication token denoted by Γ, while I_{DAA} calculates a certificate for the TPM's DAA Public Key PK. Let $x_0, \ldots, x_n$ be elements from $\mathbb{Z}_q$ which encodes attributes belonging to the signer (x_0 corresponds to the TPM's secret key and $x_0 G_0$ is the TPM's public key). The authentication token is then a discrete logarithm representation: $\Gamma = \sum_{k=0}^{k=n} x_k G_k$.

I_{DAA}'s signing secret key consists of two integers $x, y \in \mathbb{Z}_q$. $\tilde{X} = x\tilde{G}_0$ and $\tilde{Y} = y\tilde{G}_0$ correspond to I_{DAA}'s public key. Let $u, v \in \mathbb{Z}_q$ be the VC issuer's private key. $\tilde{U} = u\tilde{G}_0$ and $\tilde{V} = v\tilde{G}_0$ correspond to the VC issuer's public key. I_{DAA} proves that their key is well formed and registers the key and proof as $\left((\tilde{X}, \tilde{Y}), (x, y), \pi_{ipk}^{DAA}\right)$. Similarly, I_{VC} proves that their key is well formed and registers the key and proof as $\left((\tilde{U}, \tilde{V}), (u, v), \pi_{ipk}^{VC}\right)$.

4.3.2 Issuance of Verifiable Credentials (VCs)

The TPM chooses a secret key $x_0 \leftarrow \mathbb{Z}_q$, sets its public key $PK = x_0 G_0$, and computes $\pi^{\mathcal{M}}$ as a proof of construction of PK. $\mathcal{M}$ sends $(PK,\ \pi^{\mathcal{M}})$ to the DAA issuer I_{DAA} to request a credential.

I_{DAA} verifies $\pi^{\mathcal{M}}$ to check whether the TPM Wallet is eligible to join, i.e. the TPM Wallet DAA key has not been registered before. Upon a successful validation, I_{DAA} creates a credential on PK as follows:

- First, I_{DAA} authenticates the TPM by using the TPM's endorsement key and the TPM make_credential and activate_credential functionalities. The details of this step can be found in the TPM 2.0 specifications [22].
- I_{DAA} chooses a random $r \in_R \mathbb{Z}_q$, calculates $A = rG_0, B = yA$, $C = xA + rxyPK$, $D = ryPK$. If $A = 1_{\mathbb{E}}$, where $1_{\mathbb{E}}$ represents the identity element in $\mathbb{E}$, then I_{DAA} chooses another r.
- I_{DAA} performs a Schnorr ZK proof written as $(\hat{c}, \hat{s})$, which shows that the discrete logarithms are equivalent. To do this, I_{DAA} chooses a random $\omega \in_R \mathbb{Z}_q$; and calculates the challenge

  $$\hat{c} = H_1\Big(\omega G_0 | \omega PK | \rho\Big)$$

  and $\hat{s} = \omega - \hat{c}ry$, where ρ is a message for freshness agreed by I_{DAA}, the VC issuer and the signer.
- I_{DAA} sends the PK-credential $cre_{PK} : (A, B, C, D, \hat{c}, \hat{s})$ to the TPM Wallet which represents the credential that corresponds to the TPM Wallet with embedded TPM with Public DAA Key $PK = x_0 G_0$.
- Upon receiving cre_{PK}, the Wallet verifies the credential under I_{DAA}'s public keys $\tilde{X}$ and $\tilde{Y}$ as follows:

  1. Check $\tau(A, \tilde{Y}) \stackrel{?}{=} \tau(B, \tilde{G}_0)$;
     $\tau(A + D, \tilde{X}) \stackrel{?}{=} \tau(C, \tilde{G}_0)$.
  2. Verify the discrete logarithm equivalence via $(\hat{c}, \hat{s})$:

     $$\hat{c} \stackrel{?}{=} H_1\Big(\hat{c}B + \hat{s}G_0 | \hat{c}D + \hat{s}PK | \rho\Big)$$

  3. Accept cre_{PK} if both the above verification pass; otherwise reject it.

- The Wallet sends a request to I_{VC} to issue an attribute key credential, the Wallet also sends the TPM public DAA key PK to I_{VC}.
- I_{VC} authenticates the Wallet and defines the user's attribute space U that contains the set of attributes $x_1, \ldots x_n$ that correspond to the TPM Wallet with TPM public key $PK = x_0 G_0$.
- I_{VC} sets the Wallet attribute key as follows: $Key_w = \{x_1, x_2, \ldots, x_n\}$ and sends it to the TPM Wallet.
- I_{VC} calculates $\Gamma = PK + \sum_{k=1}^{k=n} x_k G_k$ which represents the Wallet public key with an embedded TPM of public key PK.

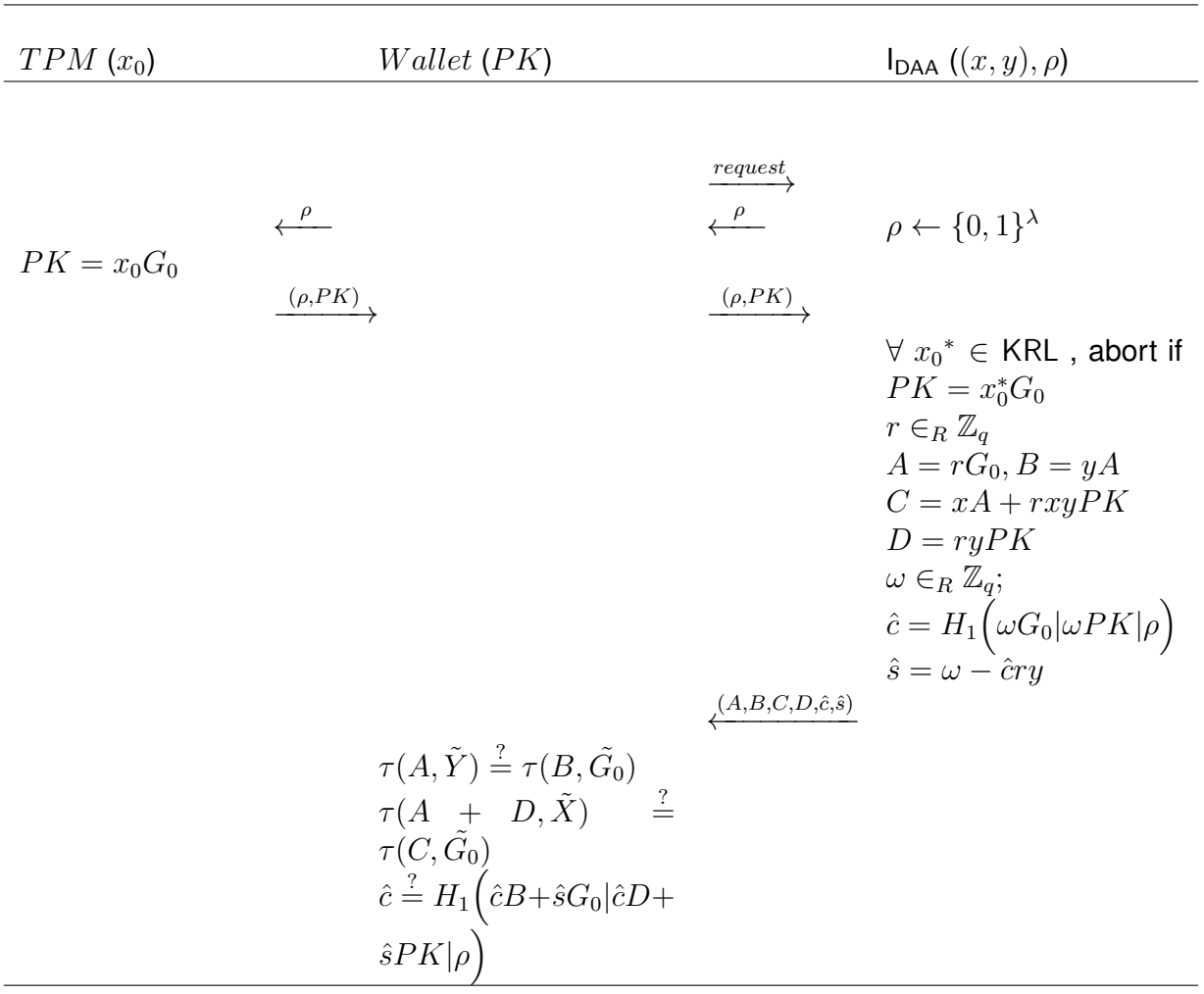

Figure 4.2: The Join Protocol with $\mathsf{I}_{\mathsf{DAA}}$

- The Wallet stores $(cre_{PK}, x_1, \ldots, x_n)$, creates a DAA signature σ^{DAA} via its TPM, and sends it to I_{VC}.
- I_{VC} verifies the Wallet's signature; if the verification passes,I_{VC} generates a verifiable credential cre_w, which is a signature on the Wallet's attribute key $Key_w = \{x_1, x_2, \ldots, x_n\}$ by performing the following steps:
  1. I_{VC} chooses a random $t \in_R \mathbb{Z}_q$, calculates $A_w = tG, B_w = vA_w$, $C_w = uA_w + tuv\Gamma$, $D_w = tv\Gamma$ and $E_{W_k} = tvG_k \; \forall \; k \in [0, n]$. If $A_w = 1_{\mathbb{E}}$, then I_{VC} chooses another t.
  2. I_{VC} chooses a random $\gamma \in_R \mathbb{Z}_q$; and calculates the challenge $\hat{c}_w = H_1\Big(\gamma G|\gamma G_0| \ldots |\gamma G_n|\gamma\Gamma|\rho\Big)$ and $\hat{s}_w = \gamma - \hat{c}_w tv$, where ρ is a message for freshness agreed by I_{VC}, I_{DAA} and the signer.
- I_{VC} sends the credential $cre_w = (A_w, B_w, C_w, D_w, E_{W_k}, \hat{c}_w, \hat{s}_w)$ to the Wallet via a secure transmission channel.
- Upon receiving cre_w, the Wallet verifies the signatures under I_{VC} public keys $\tilde{U}$ and $\tilde{V}$ as follows:
  1. Check $\tau(A_w, \tilde{V}) \stackrel{?}{=} \tau(B_w, \tilde{G_0})$;
     $\tau(A_w + D_w, \tilde{U}) \stackrel{?}{=} \tau(C_w, \tilde{G_0})$.
  2. The Wallet calculates $\hat{c'}_w = H_1\Big(\hat{c}_w B_w + \hat{s}_w G|\hat{c}_w E_{W_0} + \hat{s_w} G_0|\hat{c}_w E_{W_1} + \hat{s_w} G_1| \ldots |\hat{c}_w E_{W_n} + \hat{s}_w G_n|\hat{c}_w D_w + \hat{s}_w \Gamma|\rho\Big)$, accepts if $\hat{c'}_w = \hat{c}_w$.
- The final device credential is then $cre = (cre_{PK}, cre_w)$.

4.3.3 Self-Issuance of Verifiable Presentations (VPs)

- The DAA Bridge(host) retrieves the credential cre, aborts if no such credential exists.
- The TPM checks if the policy is satisfied, i.e the device is in correct state. If yes, the platform creates an unlinkable DAA signature σ_{DAA} using its certified public key credential cre_{PK}. The DAA Wallet creates a Proof of Knowledge (DAA signature) that it processes a valid credential for the attribute key $Key_w = (x_1, \ldots, x_n)$ and such that the overall DAA signature is only verified under a certified device public key $\Gamma = PK + \sum_{k=1}^{k=n} x_k G_k$, where PK is a certified TPM key from I_{DAA}.

  To generate a DAA-A signature σ_{DAA}, the Wallet performs as follows:
  1. **Blind**: The Wallet creates a blinding factor $a \in_{R\ q}$.
  2. The Wallet calculates: $A' = aA, B' = aB, C' = aC; D' = aD,$ and $A'_w = aA_w; B'_w = aB_w, C'_w = aC_w, D'_w = aD_w, E'_{W_k} = aE_{W_k} \; \forall \; k \in [0, n]$. The Wallet sends $(B' + E'_{W_0})$ to the TPM.
  3. **TPM Commit**: The TPM selects random $\omega_0 \in_R \mathbb{Z}_q$ and calculates $R_0 = \omega_0(B' + E'_{W_0})$, the TPM sends R_0 to the Wallet.

$TPM\ (x_0)$	$Wallet\ (PK)$	$\mathsf{I_{VC}}\ ((u,v),\rho)$
		$\xrightarrow{request_{VC}(PK)}$
		$\xleftarrow{Key_w=\{x_1,x_2,...,x_n\},\ n\ \in_R\ Z}$
σ^{DAA} on (Key_W, n)	$\xrightarrow{\sigma^{DAA}}$	$\xrightarrow{\sigma^{DAA}}$
		Verifies σ^{DAA} on (Key_W, n) $\Gamma = PK + \sum_{k=1}^{k=n} x_k G_k$ $t \in_R \mathbb{Z}_q$ $A_w = tG, B_w = vA_w\ C_w = uA_w + tuv\Gamma$ $D_w = tv\Gamma$ $E_{W_k} = tvG_k\ \forall\ k \in [0,n]$ $\gamma \in_R \mathbb{Z}_q$ $\hat{c}_w = H_1\Big(\gamma G \vert \gamma G_0 \vert \ldots \vert \gamma G_n \vert \gamma\Gamma \vert \rho\Big)$ $\hat{s}_w = \gamma - \hat{c}_w tv$
		$\xleftarrow{(A_w,B_w,C_w,D_w,E_{W_k},\hat{c}_w,\hat{s}_w,\rho)}$
	$\tau(A_w, \tilde{V}) \stackrel{?}{=} \tau(B_w, \tilde{G_0})$	
	$\tau(A_w + D_w, \tilde{U}) \stackrel{?}{=} \tau(C_w, \tilde{G_0})$	
	$\hat{c'}_w = H_1\Big(\hat{c}_w B_w + \hat{s}_w G \vert \hat{c}_w E_{W_0} + \hat{s_w} G_0 \vert \hat{c}_w E_{W_1} + \hat{s_w} G_1 \vert \ldots \vert \hat{c}_w E_{W_n} + \hat{s}_w G_n \vert \hat{c}_w D_w + \hat{s}_w \Gamma \vert \rho\Big)$	
	$\hat{c'}_w \stackrel{?}{=} \hat{c}_w$	
	$cre = (cre_{PK}, cre_w)$	

Figure 4.3: The Join Protocol with $\mathsf{I_{VC}}$

4. **Wallet Commit**: Let $\mathcal{D}$ be the set of indices of the disclosed attributes needed for a specific service, and let $\mathcal{P} = \{1, \ldots, n\} \setminus \mathcal{D}$ represents the set of indices of all other committed (hidden) attributes. We can represent $\mathcal{P}$ by the set $\{1, \ldots p\}$ which denotes the indices of the committed attributes with $p \leq n$.

   For the committed attributes, the Wallet selects random $\omega_1, \ldots, \omega_p$ from $\mathbb{Z}_q$, calculates $R_{W_k} = \omega_k E'_{W_k} \; \forall \; k \in \mathcal{P}$, and stores $\omega_1, \ldots, \omega_p$ in a protected place.
5. **Hash**: The Wallet calculates the Hash value.

$$c = H_1(A'|B'|C'|D'|A'_w|B'_w|C'_w|D'_w|E'_{W_0}|E'_{W_1}|\ldots|E'_{W_n}|R_0 + \sum_{k \in \mathcal{P}} R_{W_k}|m')$$

   for all $k \in \mathcal{P}$ and sends it to the TPM, where m' is the message to be signed.
6. **TPM-Sign**: The TPM outputs a signature $s_0 = \omega_0 + cx_0$.

- The Wallet signs each of the committed attributes, and outputs $s_k = \omega_k + cx_k \; \forall \; k \in \mathcal{P}$.
- The Wallet sends $\sigma_{DAA} = (A', B', C', D', A'_w, B'_w, C'_w, D'_w, E'_{W_0}, \ldots, E'_{W_n}, s_0, s_{k \in \mathcal{P}}, x_{k \in \mathcal{D}}, c)$ to the Verifier.

$TPM\ (x_0)$		$Wallet\ (PK, Key_W)$
		$a \in_R \mathbb{Z}_q$. $A' = aA, B' = aB, C' = aC$ $D' = aD$ $A'_w = aA_w; B'_w = aB_w$ $C'_w = aC_w, D'_w = aD_w$ $E'_{W_k} = aE_{W_k} \; \forall \; k \in [0, n]$
	$\xleftarrow{(B' + E'_{W_0})}$	
$\omega_0 \in_R \mathbb{Z}_q$ $R_0 = \omega_0(B' + E'_{W_0})$		
	$\xrightarrow{R_0}$	
		$\{\omega_1, \ldots, \omega_p\} \in_R \mathbb{Z}_q$ $R_{W_k} = \omega_k E'_{W_k} \; \forall \; k \in \mathcal{P}$ $c = H_1(A'\|B'\|C'\|D'\|A'_w\|B'_w\|C'_w\|D'_w\|E'_{W_0}\|E'_{W_1}\|\ldots\|E'_{W_n}\|$ $R_0 + \sum_{k \in \mathcal{P}} R_{W_k}\|m')$
	$\xleftarrow{c}$	
$s_0 = \omega_0 + cx_0$	$\xrightarrow{s_0}$	
		$s_k = \omega_k + cx_k \; \forall \; k \in \mathcal{P}$ $\sigma = (A', B', C', D', A'_w, B'_w C'_w, D'_w, E'_{W_0}, \ldots, E'_{W_n},$ $s_k, s_0, c)$

Figure 4.4: Creating Verifiable Presentations

$TPM\ (x_0)$		$Wallet\ (PK, Key_W)$
		$a \in_R \mathbb{Z}_q.$ $A' = aA, B' = aB, C' = aC\ D' = aD$ $A'_w = aA_w; B'_w = aB_w$ $C'_w = aC_w, D'_w = aD_w$ $E'_{W_k} = aE_{W_k}\ \forall\ k \in [0, n]$
	$\xleftarrow{\texttt{TPM2\_Commit}(B' + E'_{W_0})}$	
$\omega_0 \in_R \mathbb{Z}_q$ $R_0 = \omega_0(B' + E'_{W_0})$		
	$\xrightarrow{R_0}$	
		$\{\omega_1, \ldots, \omega_p\} \in_R \mathbb{Z}_q$ $R_{W_k} = \omega_k E'_{W_k}\ \forall\ k \in \mathcal{P}$ $c = H_1(A'\|B'\|C'\|D'\|A'_w\|B'_w\|C'_w\|D'_w\|E'_{W_0}\|E'_{W_1}\|\ldots\|E'_{W_n}\|$ $R_0 + \sum_{k \in \mathcal{P}} R_{W_k}\|m')$
	$\xleftarrow{\texttt{TPM2\_Sign}(c)}$	
$s_0 = \omega_0 + cx_0$	$\xrightarrow{s_0}$	
		$s_k = \omega_k + cx_k\ \forall\ k \in \mathcal{P}$ $\sigma = (A', B', C', D', A'_w, B'_w C'_w, D'_w, E'_{W_0}, \ldots, E'_{W_n},$ $s_k, s_0, c)$

Figure 4.5: Creating Verifiable Presentations

4.3.4 Verification of Verifiable Presentations (VPs)

- The Verifier checks the attributes and verifies the DAA signature as follows:

  1. Verify the modified CL certificate,
  2. Check $\tau(A', \tilde{Y}) \stackrel{?}{=} \tau(B', \tilde{G}_0)$;
     $\tau(A' + D', \tilde{X}) \stackrel{?}{=} \tau(C', \tilde{G}_0)$.
  3. Check $\tau(A'_w, \tilde{V}) \stackrel{?}{=} \tau(B'_w, \tilde{G}_0)$;
     $\tau(A'_w + D'_w, \tilde{U}) \stackrel{?}{=} \tau(C'_w, \tilde{G}_0)$.
  4. Verify the discrete logarithm equivalence by the batch proof trick: $t_0, t_1, \ldots, t_n \in$;

     $$\tau(t_0 E'_{W_0} + \ldots + t_n E'_{W_n}, \tilde{G}) \stackrel{?}{=} \tau(B'_w, t_0\tilde{G}_0 + \ldots + t_n\tilde{G}_n)$$

  5. Verify the Schnorr ZK proof of knowledge of the hidden attributes:
     $\mu_W = \sum_{k\in\mathcal{P}} s_k E'_{W_k} + s_0(B' + E'_{W_0}) - c(D' + D'_w - \sum_{k\in\mathcal{D}} x_k E'_{W_k})$
     $c \stackrel{?}{=} H_1(A'|B'|C'|D'|A'_w|B'_w|C'_w|D'_w|E'_{W_0}|\ldots|E'_{W_n}|\mu_W|m')$

- If the verification pass, the Verifier creates an *Audit Token* by creating a signature $\sigma_V \leftarrow Sign_V(\sigma_{DAA}, x_k)$ where x_k are the set of disclosed attributes with indices $k \in \mathcal{D}$.
- Outputs $(\sigma_V, \sigma_{DAA}, x_i)$

$Wallet$ (Key_W)	Verifer ($(\tilde{X}, \tilde{Y})$ $(\tilde{U}, \tilde{V})$)
$\xrightarrow{(s_0, s_{k\in\mathcal{P}} x_{k\in\mathcal{D}}, c)}$	$\tau(A', \tilde{Y}) \stackrel{?}{=} \tau(B', \tilde{G}_0)$ $\tau(A' + D', \tilde{X}) \stackrel{?}{=} \tau(C', \tilde{G}_0)$ $\tau(A'_w, \tilde{V}) \stackrel{?}{=} \tau(B'_w, \tilde{G}_0)$ $\tau(A'_w + D'_w, \tilde{U}) \stackrel{?}{=} \tau(C'_w, \tilde{G}_0)$ $t_0, t_1, \ldots, t_n \in_R$ $\tau(t_0 E'_{W_0} + \ldots + t_n E'_{W_n}, \tilde{G}) \stackrel{?}{=} \tau(B'_w, t_0\tilde{G}_0 + \ldots + t_n\tilde{G}_n)$ $\mu_W = s_0(B' + E'_{W_0}) + \sum_{k\in\mathcal{P}} s_k E'_{W_k} - c(D' + D'_w - \sum_{k\in\mathcal{D}} x_k E'_{W_k})$ $c \stackrel{?}{=} H_1(A'\|B'\|C'\|D'\|A'_w\|B'_w\|C'_w\|D'_w\|E'_{W_0}\|\ldots\|E'_{W_n}\|\mu_W\|m')$

Figure 4.6: Verification

4.4 Computation Costs

We compare the computational efficiency of our scheme with the other schemes proposed in the literature. In particular, we show the **computational cost for the TPM in the sign algorithm**, for **the host in the sign algorithm**, and for **the Verifier in the verify algorithm**, as these are the algorithms that will be used frequently.

We denote k exponentiation in group $\mathbb{E}$ by $k\mathbb{E}$, and $k\tau$ denotes k pairing operations. Note that the schemes proposed in [7, 22] have extra link tokens. Having no linking requirement in our scheme has significantly reduced the computational costs for signing and verification, thus **we omit the computations of the link tokens from our comparison** (we only keep the link token corresponding to the TPM attribute x_0 as this is a part of the TPM's signature in other schemes). Our **TPM signing operation** is the most efficient, as it only needs one operation in $\mathbb{E}$. The **host signing** is as efficient as the CL- based scheme in [7], which is the most efficient signing between all schemes. Our signing beats [22], where their signing part requires an operation in $\tilde{\mathbb{F}}$ and two parings.

In [23], no concrete description of the proposed RDF proof was provided, thus we cannot tell the exact computation costs of π^{RDF} for the signing and verification. However, as the BBS+ signature is followed in the aforementioned scheme for issuing the credentials, we include only the BBS+ computational cost for signing and verification. Table 4.1 shows that our signing scheme is more efficient than [23], even when including only the BBS+ computational cost. As we achieve the **VC-user key binding security property**, which is not achieved in any other scheme in the literature, two extra pairings for verifying the signature are added. This is intended to bind the DAA issuer (that certifies the hardware (TPM) key) to the VC Issuer (that issues the Verifiable Credential that is bound to the TPM key). This slightly increases the computational cost of the verification in our scheme. However, it is important to note that **we provide a rigorous proof that captures the VC-user key binding security property that was not addressed in any other scheme**.

Scheme	TPM Sign	Host Sign	Verify
[7] (CL)	$3\mathbb{E}$	$(4+n+\mathcal{P})\mathbb{E}$	$(2+3n)\mathbb{E}+3\tau$
[7] (SDH)	$3\mathbb{E}$	$(4+\mathcal{P})\mathbb{E}+2\tau$	$(6+n)\mathbb{E}+2\tau$
[3] (BBS+)	$3\mathbb{E}$	$(3+\mathcal{P})\mathbb{E}+\tilde{\mathbb{F}}+1\tau$	$(8+n)\mathbb{E}+1\tau$
[23] (BB+ and RDF)	-	$(3+\mathcal{P})\mathbb{E}+\tilde{\mathbb{F}}+1\tau$ + π^{RDF}_{sign}	$(8+n)\mathbb{E}+1\tau$ + π^{RDF}_{verify}
Our scheme	$1\mathbb{E}$	$(5+n+\mathcal{P})\mathbb{E}$	$(4+3n)\mathbb{E}+5\tau$

Table 4.1: Computational Cost Comparison

4.5 Security Model

Here, we explain the interfaces of the proposed protocol ideal functionality $\mathcal{F}$ in the UC framework based on the functionality originally proposed in [4] and later adopted in [6,17]. The UC framework allows us to focus the analysis on a single protocol instance with a globally unique session identifier sid. $\mathcal{F}$ uses two session identifiers of the form $\mathsf{sid}_{\mathsf{DAA}} = (\mathsf{I}_{\mathsf{DAA}}, \mathsf{sid}')$ and $\mathsf{sid}_{\mathsf{VC}} = (\mathsf{I}_{\mathsf{VC}}, \mathsf{sid}'')$ for some DAA aand VC issuers $\mathsf{I}_{\mathsf{DAA}}$ and I_{VC} and unique strings $\mathsf{sid}', \mathsf{sid}''$. The restriction that **the issuers' identity must be included in the session identifiers** $\mathsf{sid}_{\mathsf{DAA}}, \mathsf{sid}_{\mathsf{VC}}$ guarantees that **each issuer can initialize its own instance of the functionality**. In real-world scenarios, these strings are m**apped to the issuer public key**, and all parties use the sid to link their stored key material to the particular issuer. We define $JOIN$, and $SIGN$ sub-session identifiers

jsid_{DAA}, jsid_{VC} and $ssid$, respectively, to distinguish several join and sign sessions that might run in parallel.

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

We assume that the host $\mathcal{H}$ represents the Wallet. Thus, $\mathcal{M}$ and $\mathcal{H}$ correspond to the TC and the Wallet respectively. A parameter $\mathbb{P}$ is used to describe which proofs over the attributes platforms can make. This generic approach adopted from [3] lets the functionality $\mathcal{F}$ to capture both simple protocols that only support selective disclosure and more advanced protocols that support arbitrary predicates. Let $attr = \{a_1...a_L\}$ represents a set of attributes a_i, where $i \in [1, L]$ for some integer L, then every element $p \in \mathbb{P}$ represents a predicate over the attributes defined by $p : attr \rightarrow \{0, 1\}$, i.e $p(attr) = 1$ implies that a device has all attributes in $attr$, $p(attr) = 0$ otherwise.

4.5.1 Ideal Functionality Algorithms

We define the algorithms that will be used inside the functionality $\mathcal{F}$ as follows:

`Kgen`(1^β): A probabilistic algorithm that takes a security parameter β for honest platforms. It generates the DAA signing key for honest TPM.

`Sign`: A probabilistic algorithm that runs $\text{sig}_{\text{DAA-A}}(tsk, p, m)$ to create signatures on behalf of honest TPMs. First, it creates CL signatures (DAA and VC credentials) using both Issuers' keys on tsk and the attribute values where the disclosed attributes are taken from p and the undisclosed attributes are set to dummy values. Next, the algorithm performs the real world signing algorithm (performing both the tasks for the TC Bridge (host) and the TPM).

`Verify`: A deterministic algorithm that outputs a binary result on signature correctness. It uses the sub-algorithm $\text{ver}_{\text{DAA-A}}(\sigma_{DAA-A}, m, p)$ which for a signature σ_{DAA-A}, a message m, and a predicate over the attributes, it outputs $f = 1$ if the signature is valid; $f = 0$ otherwise.

`Identify`: A deterministic algorithm will be used to ensure consistency with the ideal functionality $\mathcal{F}$'s internal records, by connecting a signature to the key used to generate it. It runs the following sub-algorithm $\text{identify}_{\text{DAA-A}}(tsk, \sigma_{DAA-A}, m)$: It outputs 1 if a key tsk was used to produce a signature σ_{DAA-A}, and 0 otherwise.

4.6 Analysis of the Security model

Security in the UC framework follows the simulation-based paradigm, where a protocol is secure when it is as secure as an ideal functionality that performs the desired tasks in a way that is secure by design. We define our DAA-A ideal functionality $\mathcal{F}$, which covers the following high-level security properties:

SETUP

- On input ($\mathsf{SETUP}_{\mathsf{DAA}}, \mathsf{sid}_{\mathsf{DAA}}$) from the DAA-issuer $\mathsf{I}_{\mathsf{DAA}}$, output ($\mathsf{SETUP}_{\mathsf{DAA}}$, $\mathsf{sid}_{\mathsf{DAA}}$) to $\mathcal{S}$.
- On input ($\mathsf{SETUP}_{\mathsf{VC}}, \mathsf{sid}_{\mathsf{VC}}$) from the VC-issuer I_{VC}, output ($\mathsf{SETUP}_{\mathsf{VC}}$, $\mathsf{sid}_{\mathsf{VC}}$) to $\mathcal{S}$.
- **Set Algorithms**: On input (ALG, $\mathsf{sid} = (\mathsf{sid}_{\mathsf{DAA}}, \mathsf{sid}_{\mathsf{VC}})$, Sign, Verify, Identify, Kgen) from $\mathcal{S}$,
- Check that Verify and Identify are deterministic. [CHECK I]
- Store the algorithms and output ($\mathsf{SETUPDONE}, \mathsf{sid}$) to $\mathsf{I}_{\mathsf{DAA}}$ and I_{VC} respectively.

DAA-JOIN

- On input $(\mathsf{JOIN}_{\mathsf{DAA}}, \mathsf{sid}, \mathsf{jsid}_{\mathsf{DAA}}, \mathcal{M})$ from host $\mathcal{H}$, create a join session record $\langle \mathsf{jsid}_{\mathsf{DAA}}, \mathcal{M}, \mathcal{H} \rangle$ and output ($\mathsf{JOINPROCEED}_{\mathsf{DAA}}, \mathsf{sid}, \mathsf{jsid}_{\mathsf{DAA}}, \mathcal{M}$) to $\mathsf{I}_{\mathsf{DAA}}$.
- On input $(\mathsf{JOINPROCEED}_{\mathsf{DAA}}, \mathsf{sid}, \mathsf{jsid}_{\mathsf{DAA}})$ from $\mathsf{I}_{\mathsf{DAA}}$, output $(\mathsf{JOINCOMPLETE}_{\mathsf{DAA}}, \mathsf{sid}, \mathsf{jsid}_{\mathsf{DAA}})$ to $\mathcal{S}$.
- On input $(\mathsf{JOINCOMPLETE}_{\mathsf{DAA}}, \mathsf{sid}, \mathsf{jsid}_{\mathsf{DAA}}, tsk)$ from $\mathcal{S}$.
- Abort if $\mathsf{I}_{\mathsf{DAA}}$ or $\mathcal{M}$ is honest and a record $\langle \mathcal{M}, *, * \rangle \in \mathsf{MemberList}_{\mathsf{DAA}}$ already exists. [CHECK II]
- If $\mathcal{M}$ and $\mathcal{H}$ are honest, set $tsk \leftarrow \perp$.
- Else, verify that the provided tsk is eligible by checking:
- $\mathsf{CheckkeyHonest}(tsk) = 1$ if $\mathcal{M}$ is honest and $\mathcal{H}$ is corrupt, [CHECK III] or
- $\mathsf{CheckkeyCorrupt}(tsk) = 1$ if $\mathcal{M}$ is corrupt and $\mathcal{H}$ is honest. [CHECK IV]
- Insert $\langle \mathcal{H}, \mathcal{M}, tsk \rangle$ into $\mathsf{MemberList}_{\mathsf{DAA}}$ and output ($\mathsf{JOINED}_{\mathsf{DAA}}, \mathsf{sid}, \mathsf{jsid}_{\mathsf{DAA}}$) to $\mathcal{H}$.

VC-JOIN

- On input $(\mathsf{JOINED}_{\mathsf{DAA}}, \mathsf{sid}, \mathcal{M})$ from host $\mathcal{H}$, create a join session record $\langle \mathsf{jsid}_{\mathsf{VC}}, \mathcal{M}, \mathcal{H} \rangle$ and output ($\mathsf{JOINPROCEED}_{\mathsf{VC}}, \mathsf{sid}, \mathsf{jsid}_{\mathsf{VC}}, \mathcal{M}$) to I_{VC}.
- On input $(\mathsf{JOINPROCEED}_{\mathsf{VC}}, \mathsf{sid}, \mathsf{jsid}_{\mathsf{VC}}, attr)$ from I_{VC}, output $(\mathsf{JOINCOMPLETE}_{\mathsf{VC}}, \mathsf{sid}, \mathsf{jsid}_{\mathsf{VC}}, attr')$ to $\mathcal{S}$, where $attr = \perp$ if $\mathcal{M}$ and $\mathcal{H}$ are honest and $attr' \leftarrow attr$ otherwise.
- On input $(\mathsf{JOINCOMPLETE}_{\mathsf{VC}}, \mathsf{sid}, \mathsf{jsid}_{\mathsf{VC}}, tsk, attr)$ from $\mathcal{S}$.
  - Abort if $\mathsf{I}_{\mathsf{DAA}}$ or $\mathcal{M}$ is honest and there is no record $\langle \mathcal{M}, tsk, * \rangle$ in $\mathsf{MemberList}_{\mathsf{DAA}}$. [CHECK V]
  - Abort if I_{VC} or $\mathcal{M}$ is honest and a record $\langle \mathcal{M}, *, * \rangle \in \mathsf{MemberList}_{\mathsf{VC}}$ already exists. [CHECK VI]
  - Insert $\langle \mathcal{H}, \mathcal{M}, tsk, attr \rangle$ into $\mathsf{MemberList}_{\mathsf{VC}}$ and output ($\mathsf{JOINED}_{\mathsf{VC}}$, $\mathsf{sid}_{\mathsf{DAA}}$, $\mathsf{jsid}_{\mathsf{VC}}$) to $\mathcal{H}$.

Figure 4.7: The Setup and Join interfaces of $\mathcal{F}$

SIGN

- On input (SIGN, sid, ssid, $\mathcal{M}, m, p$) from $\mathcal{H}$. If $\mathcal{H}$ is honest and no entry $\langle \mathcal{M}, \mathcal{H}, tsk \rangle$ exists in MemberList$_{\text{DAA}}$ and $\langle \mathcal{M}, \mathcal{H}, tsk, attr \rangle$ such that $p(attr) = 1$ in MemberList$_{\text{VC}}$, abort. Else, Create a sign session record $\langle$ssid, $\mathcal{M}, \mathcal{H}, m\rangle$ and output (SIGNPROCEED, sid, ssid, m) to $\mathcal{M}$.
- On input (SIGNPROCEED, sid, ssid) from $\mathcal{M}$. Output (SIGNCOMPLETE, sid, ssid) to $\mathcal{S}$.
- On input (SIGNCOMPLETE, sid, ssid, σ_{DAA-A}) from $\mathcal{S}$.
  - If I$_{\text{VC}}$ is honest, check that ($\mathcal{M}, \mathcal{H}, *, p$) with $p(attr) = 1$ exists in MemberList$_{\text{VC}}$. [CHECK VII]
  - If $\mathcal{M}$ and $\mathcal{H}$ are honest, ignore the adversary's signature and internally generate the signature for a fresh or established tsk:
    - Find tsk from $(\mathcal{M}, tsk) \leftarrow$ DOMAIN KEYS LIST. If no such tsk exists, set $tsk \leftarrow$ Kgen() and check CheckkeyHonest$(tsk) = 1$. [CHECK VIII]
    - Compute signature as: $\sigma_{DAA-A} \leftarrow$ sig$_{\text{DAA-A}}(tsk, p, m)$.
    - Check ver$_{\text{DAA-A}}(\sigma_{DAA-A}, m, p) = 1$. [CHECK VIX]
    - Check:identify$_{\text{DAA-A}}(\sigma_{DAA-A}, m) = 1$ [CHECK X] and check that there is no $\mathcal{M}' \neq \mathcal{M}$ with $p(attr) = 1$ registered in DOMAIN KEYS LIST with: identify$_{\text{DAA-A}}(\sigma_{DAA-A}, m) = 1$. [CHECK XI]
  - If $\mathcal{M}$ is honest, store $\langle \sigma_{DAA-A}, m, p, \mathcal{M}, \mathcal{H} \rangle$ in SIGNED LIST.
  - Output (SIGNATURE, sid, ssid, σ_{DAA-A}) to $\mathcal{H}$.

VERIFY

- On input (VERIFY, sid, m, p, σ_{DAA-A}, KRL) from some party $\mathcal{V}$.
- Retrieve all pairs $(tsk, \mathcal{M})$ from $\langle \mathcal{M}, *, tsk \rangle \in$ DOMAIN KEYS LIST where: identify$_{\text{DAA-A}}(\sigma_{DAA}, m)$ = 1. Set $f = 0$ if at least one of the following conditions holds:
  - More than one key tsk was found. [CHECK XII]
  - I$_{\text{DAA}}$ is honest and no pair $(tsk, \mathcal{M})$ was found. [CHECK XIII]
  - I$_{\text{VC}}$ is honest and no $(tsk, p, \mathcal{M})$ was found with with $p(attr) = 1$. [CHECK XIV]
  - There is an honest $\mathcal{M}$ but no entry $\langle *, m, p, \mathcal{M} \rangle \in$ SIGNED LIST exists. [CHECK XV]
  - There is a $tsk^* \in$ KRL where identify$_{\text{DAA}}(\sigma_{DAA}, m, tsk^*) = 1$ and no pair $(tsk, \mathcal{M})$ for an honest $\mathcal{M}$ was found. [CHECK XVI]
  - If $f \neq 0$, set f = ver$_{\text{DAA-A}}(\sigma_{DAA-A}, m)$. [CHECK XVI]
  - Add $\langle \sigma_{DAA-A}, m,$ KRL, $f \rangle$ to VERIFIED LIST, output (VERIFIED, sid, f) to $\mathcal{V}$.

Figure 4.8: The Sign and Verify interfaces of $\mathcal{F}$

SP1: DAA-VC credential Binding: If at least one of the issuers is honest, a verifiable credential can only be issued by the VC-Issuer to the TPM-Wallet, which has a unique TPM's key tsk registered in $\mathsf{MemberList}_{\mathsf{DAA}}$. This is achieved in CHECK V. Moreover, the issued Verifiable Credential is bound to a unique TPM key, i.e, no adversary can create a Verifiable Presentation using a TPM key that is not bound to its Verifiable Credential. At the end of VC-Join, a new member $\langle \mathcal{H}, \mathcal{M}, tsk, attr \rangle$ is added into $\mathsf{MemberList}_{\mathsf{VC}}$. This membership binds the tsk of the TPM to the corresponding approved attribute set $attr$ issued for the Wallet. Whenever the Wallet requires a signature, a check that this Wallet has a record in $\mathsf{MemberList}_{\mathsf{VC}}$ (CHECK VII) is performed.

SP2: Wallet Correct State: A Wallet is allowed to use a DAA key for signing a Verifiable Credential or creating a Verifiable Presentation only if no compromise has been detected.

SP3: Full Anonymity: An adversary that is given two signatures cannot distinguish whether both signatures were created by one honest TPM-based Wallet or not. Also, no adversary can link signatures on two messages m_1 and m_2, even if these signatures were created by the same honest signer. Full anonymity of signatures created by an honest TPM $\mathcal{M}$ together with an honest host $\mathcal{H}$ is guaranteed by $\mathcal{F}$ due to the random choice of tsk for every signature. This guarantees identity anonymity of the Wallet. Also, on input $(\mathsf{JOINPROCEED}_{\mathsf{VC}}, \mathsf{sid}, \mathsf{jsid}_{\mathsf{VC}}, attr)$ from I_{VC}, $\mathcal{F}$ outputs $(\mathsf{JOINCOMPLETE}_{\mathsf{VC}}, \mathsf{sid}, \mathsf{jsid}_{\mathsf{VC}}, attr')$ to $\mathcal{S}$, where $attr'$ (same size as the original attribute set $attr$) is a dummy attribute set that is assigned to honest Wallets. This guarantees that the signature leaks no information (perfectly hides) the committed attributes in the real-world protocol, but only reveals the number of total attributes. The Full anonymity property must hold even when the $\mathsf{I}_{\mathsf{DAA}}$ or/and I_{VC} are corrupt.

SP4: Key Uniqueness: When storing a new tsk, the functionality $\mathcal{F}$ checks if $\mathsf{CheckkeyHonest} = 1$ or $\mathsf{CheckkeyCorrupt} = 1$. If the device is corrupt, $\mathcal{F}$ checks that $\mathsf{CheckkeyCorrupt} = 1$ for the tsk that was extracted by the simulator from $\pi^{\mathcal{M}}$. This check prevents the adversary from choosing different keys that match the same signature.

SP5: Correctness: If the signer and Verifier are honest, this implies $tsk^* \notin$ KRL, and the signatures generated by the signer will be accepted by the Verifier with overwhelming probability. This is achieved in our security model by CHECK X, which guarantees that each signature will always trace to the signer who created it. CHECK XVI also guarantees that the provided signature will not match any revoked key to insure correct revocation. The VC Issuer is in charge of the attributes, and must explicitly allow a Wallet to be issued certain attributes in the $\mathsf{JOINPROCEED}_{\mathsf{VC}}$. The verification interface now checks whether the signer has the correct attributes, fulfilling the attribute predicate (CHECK XIV). This guarantees that no Wallet can create valid signatures with respect to attribute predicates that do not hold for the attributes of this Wallet.

SP6: Unforgeability: At least one of the issuers ($\mathsf{I}_{\mathsf{DAA}}$ or I_{VC}) should be honest when aiming at unforgeability. The unforgeability property states that no adversary can create a DAA-A signature on a message m on behalf of the TPM when no TPM signed m. By CHECK XIV, the signature must trace to one TPM tsk. If the TPM is honest, (CHECK XV) will reject any signature on messages not signed by that TPM. We highlight that our Unforgeability notion is stronger than the normal VC schemes with only one issuer, where the DAA Issuer can forge credentials and hence creates forged signatures. Our definition of unforgeability does not allow the DAA Issuer alone to forge signatures, nor the VC issuer alone, unless

they are both corrupt. This provides stronger unforgeability guarantees (i.e., if at least one check CHECK XIII or CHECK XIV doesn't hold, the signature will not successfully verify in our security model).

4.7 Security Proof

4.7.1 High level Description of the Security Proof

We start with the real-world protocol execution in Game 1. In the next game, we construct one entity C that **runs the real-world protocol for all honest parties**. Then we split C into two pieces, an **ideal functionality** $\mathcal{F}$ and a **simulator** $\mathcal{S}$ that simulates the real world parties. Initially, we start with an "empty" functionality $\mathcal{F}$. With each game, we gradually change $\mathcal{F}$ and update $\mathcal{S}$ accordingly, moving from the real world to the ideal world, and culminating to the **full ideal functionality** $\mathcal{F}$ being realized as part of the ideal world, thus proving our proposed security model presented in Section 4.5. The endmost goal of our proof is to **prove the indistinguishability between Game 1 and Game 14**, i.e., **between the complete real world and the fully functional ideal world**. This is done by proving that each game is indistinguishable from the previous one starting from Game 1 to reach Game 14.

As aforementioned, our proof starts with **setting up real-world games** (Game 1 and Game 2), followed by **introducing the ideal functionality** in Game 3. At this stage, the ideal functionality $\mathcal{F}$ only forwards its inputs to the simulator who simulates the real world. From Game 4 onward, $\mathcal{F}$ starts **executing the setup interface** on behalf of the Issuer. Moving on to Game 5, $\mathcal{F}$ handles simple **verification and identification checks** without performing any detailed checks at this stage; i.e., it only checks if the device belongs to a revocation list separately. In Games 6-7, $\mathcal{F}$ **executes the Join interface while performing checks** to keep the consistency of registered keys. It also adds **checks that allow only the devices that have successfully been enrolled to create signatures**. Game 8 proves the **anonymity of our protocol** by letting $\mathcal{F}$ handle the sign queries on behalf of honest devices by creating signatures using freshly generated random keys instead of running the signing algorithm using the device's signing key. At the end of this game, we prove that by relying on DDH and DL constructions, **an external environment will notice no change from previous games where the real-world algorithm was executed**. Now moving to Games 9 - 14, we let $\mathcal{F}$ to perform all other checks that are explained in the real world in Section 4.5). We use the "$\approx$" sign to express games' indistinguishability.

4.7.2 Security Proof of our Scheme in the UC Model

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

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

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

Game 4 ($\mathcal{F}$ handles the setup of the real-world) now behaves differently in the setup interface, as it stores the algorithms defined in Section 4.5.1. $\mathcal{F}$ also performs checks and ensures that the structure of $\mathsf{sid}_{\mathsf{DAA}}$ and $\mathsf{sid}_{\mathsf{VC}}$, which represents the DAA and the VC issuers' unique session identifiers respectively, is correct for an honest $\mathsf{I}_{\mathsf{DAA}}$ and I_{VC}, and aborts if not. When $\mathsf{I}_{\mathsf{DAA}}$ and I_{VC} are honest, $\mathcal{S}$ will start simulating them. Since $\mathcal{S}$ is now running the Issuers, it knows both issuers' secret keys. In case $\mathsf{I}_{\mathsf{DAA}}$ or/and I_{VC} is/are corrupt, $\mathcal{S}$ extracts $\mathsf{I}_{\mathsf{DAA}}$ or/and I_{VC}'s secret key from π_{ipk}^{DAA} and/or π_{ipk}^{VC}, then proceeds to the setup interface on behalf of the corrupt issuer(s). By the simulation soundness of π_{ipk}^{DAA} and π_{ipk}^{VC} , this game transition is indistinguishable for the adversary (Game 4 $\approx$ Game 3).

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

Game 6 ($\mathcal{F}$ handles the join/SP1: DAA-VC credential Binding): The join interface of $\mathcal{F}$ is now changed. Specifically, $\mathcal{F}$ stores the members that joined in its records. If $\mathsf{I}_{\mathsf{DAA}}$ and I_{VC} are honest, then $\mathcal{F}$ stores the secret keys tsk extracted from $\pi^{\mathcal{M}}$ by $\mathcal{S}$ for corrupt Wallets, respectively. $\mathcal{S}$ already knows $tsk_1, \ldots, tsk_n$ (that correspond to the attribute keys $x_1, \ldots x_n$ in the real world) because it is simulating I_{VC}. $\mathcal{F}$ sets the tracing key for each Wallet (as it already knows its key tsk) or calculates it from the extracted tsk (for corrupt Wallets).

The cases where the TPM-Wallet is already registered in $\mathsf{MemberList}_{\mathsf{DAA}}$ and $\mathsf{MemberList}_{\mathsf{VC}}$, $\mathcal{F}$ will abort the protocol. $\mathcal{F}$ will also abort if the TPM-Wallet request a VC-credential but is not registered in $\mathsf{MemberList}_{\mathsf{DAA}}$. However, $\mathsf{I}_{\mathsf{DAA}}$ and I_{VC} have already tested these cases before continuing with the query $\mathsf{JOINPROCEED}_{\mathsf{DAA}}$ and $\mathsf{JOINPROCEED}_{\mathsf{VC}}$ respectively, thus, $\mathcal{F}$ will not abort.

Knowing $tsk, tsk_1, \ldots, tsk_n$, $\mathcal{F}$ will proceed with adding the the TPM-Wallets in $\mathsf{MemberList}_{\mathsf{DAA}}$ and $\mathsf{MemberList}_{\mathsf{VC}}$. Therefore $\mathcal{F}$ and $\mathcal{S}$ can interact to simulate the real world protocol in all cases. Due to the simulation soundness of $\pi^{\mathcal{M}}$, Game 6 $\approx$ Game 5.

Game 7 ($\mathcal{F}$ further handles the join/SP2: Wallet Correct State): If $\mathsf{I}_{\mathsf{DAA}}$ and I_{VC} are honest, then $\mathcal{F}$ only allows the devices that joined and in correct state to sign. An honest Wallet will always check whether it joined with a TC in the real world protocol, so there is no difference for honest Wallets. In the case that an honest $\mathcal{M}$ performs a join protocol with a corrupt Wallet and honest Issuers, $\mathcal{S}$ will make a join query with $\mathcal{F}$, to ensure that $\mathcal{M}$ and the Wallet are in $\mathsf{MemberList}_{\mathsf{DAA}}$ and $\mathsf{MemberList}_{\mathsf{VC}}$. Also, only Wallets with the correct configuration and certified public keys can create signatures. This will be checked by the $\mathcal{F}$. Therefore, Game 7 $\approx$ Game 6.

Game 8 ($\mathcal{F}$ handles the sign/ SP3: Full Anonymity) (Simulating the TPM Wallet without knowing its secret): In this game, $\mathcal{F}$ creates anonymous signatures for honest Wallets by running the algorithms defined in the setup interface. Let us start by defining Game $7.t.t'$. In this game, $\mathcal{F}$ handles the first t' signing inputs of $\mathcal{M}_t$ for some integer t, and subsequent inputs are then forwarded to $\mathcal{S}$. For $j < t$, $\mathcal{F}$ handles all the signing queries with $\mathcal{M}_j$ using algorithms defined in Section 4.5.1. For $j > t$, $\mathcal{F}$ forwards all signing queries with $\mathcal{M}_j$ to $\mathcal{S}$, who creates signatures as before. Next, from the definition of Game $7.t.t'$, we note that Game 7.0.0=Game 6. For increasing t', Game $7.t.t'$ will eventually become equal to Game $7.t + 1.0$. This is because there can only be a polynomial number of signing queries to be processed. Therefore, for large enough t and t', $\mathcal{F}$ handles all the signing queries of all TPM's, and Game 7 is indistinguishable from Game $7.t.t'$.

Next, we want to prove that Game $7.t.t' + 1$ is indistinguishable from Game $7.t.t'$ (i.e., we want to prove that an external environment cannot distinguish when $\mathcal{F}$ internally handles the signing queries instead of merely forwarding them to $\mathcal{S}$ for a TC $\mathcal{M}_t$). Suppose an environment can distinguish a signature (created by an honest Wallet with a secret signing key tsk) from a signature constructed by the same party but with a randomly chosen fresh tsk. Then we can use that environment to break a Decisional Diffie–Hellman DDH instance α, β, γ by simulating the JOIN protocol: The first signature can be simulated using the unknown $\log_{G_0}(\alpha)$ as tsk, while for the second signature we ca use the unknown $\log_{\beta}(\gamma)$ as tsk. In the reduction, we have to be able to simulate the TPM without knowing the real TPM's tsk, but merely based on $\alpha = tskG_0$. A TPM uses tsk to set $Q \leftarrow tskG_0$ in the JOIN protocol; In the simulation, we set $Q \leftarrow \alpha$ and we simulate the proof $\pi^{\mathcal{M}}$. It is easy to see that an external environment cannot distinguish between an honest Wallet signature from a signature, by the same party, but with a randomly chosen fresh tsk in every signature (Game 8 $\approx$ Game 7, except for a negligible probability).

Game 9 (SP4: Key Uniqueness): When storing a new tsk, $\mathcal{F}$ checks if $\mathsf{CheckkeyHonest} = 1$ or $\mathsf{CheckkeyCorrupt} = 1$. If the device is corrupt, $\mathcal{F}$ checks that $\mathsf{CheckkeyCorrupt} = 1$ for tsk that the simulator extracted. This check prevents the adversary from choosing different keys. There exists only a single tsk for every valid signature where $\mathsf{identify}_{\mathsf{D}}(\sigma, m, tsk) = 1$, therefore this check will never fail due to the soundness of the proof $\pi^{\mathcal{M}}$. For honest TPM keys, $\mathcal{F}$ verifies that $\mathsf{CheckkeyHonest} = 1$ whenever it receives or generates a new key. With these checks, we avoid the registration of keys for which matching signatures already exist. Since keys for honest TPMs are chosen uniformly from an exponentially large group and every signature has exactly one matching key, the chance that a signature under that key already exists is negligible. $\mathcal{F}$ ensures, by using its internal records $\mathsf{MemberList}_{\mathsf{DAA}}$ and $\mathsf{MemberList}_{\mathsf{VC}}$, that honest users do not share the same Wallet key Γ. This is reduced with a non-negligible probability of solving the DL problem in the real world. Suppose that two honest Wallets share the same Γ. If the Wallets have different attribute sets $\{x_1, \ldots x_k\}$ and $\{x'_1, \ldots x'_n\}$, then $\Gamma = tskG_0 + \sum_{i=1}^{k} x_i G_i = tsk'G_0 + \sum_{i=1}^{n} x'_i G_i$ which results that $tsk' - tsk = log_{G_0}(\sum_{i=1}^{k} x_i G_i - \sum_{i=1}^{n} x'_i G_i)$, therefore $tsk' - tsk$ must be the discrete log solution. If the Wallets share the same attribute set $\{x_1, \ldots x_k\}$, then this yields to $tsk = tsk'$ and therefore both Wallets are joined with the same TPM which contradicts the checks performed by the DAA Issuer $\mathsf{I}_{\mathsf{DAA}}$ in the real world.

For keys of honest devices, $\mathcal{F}$ verifies that $\mathsf{CheckkeyHonest} = 1$ whenever it receives or generates a new key. With these checks, we avoid the registration of keys for which matching signatures already exist. Since keys for honest devices are chosen uniformly from an exponentially large group and every signature has exactly one matching key, the chance that a signature under that key already exists is negligible (Game 9 $\approx$ Game 8).

Game 10 (SP5: Correctness): In this game, $\mathcal{F}$ checks that honestly generated signatures are always valid. This is true, since the $\mathsf{sig}_{\mathsf{DAA}}$ algorithm always produces signatures passing through verification checks. They also satisfy
$\mathsf{identify}_{\mathsf{DAA}}(tsk, \sigma_{DAA}, m) = 1$. $\mathcal{F}$ ensures, by using its internal records $\mathsf{MemberList}_{\mathsf{DAA}}$ and $\mathsf{MemberList}_{\mathsf{VC}}$, that honest users are not sharing the same key Γ even when having the same attributes; this is reduced with non-negligible probability to solving the DL problem. Assume that $\mathcal{F}$ receives an instance $tsk, tsk_1, \ldots tsk_n \in \mathbb{Z}_q$ of the DL problem and must answer $log_{G_i}(tsk_i)\ \forall\ i \in [0, n]$. $\mathcal{F}$ chooses an honest device and simulates its tasks using the unknown discrete logarithm as its secret key. When a $tsk, tsk_1, \ldots tsk_n$ matches one of this device's signatures, then this must be the discrete log solution, as there is only one $tsk, tsk_1, \ldots tsk_n$ matching to a certain signature (Game 10 $\approx$ Game 9).

Game 11 (SP5: Correctness of Revocation): CHECK XII ensures that there are no multiple

tsk values matching one signature. $\mathcal{F}$ also checks, with the help of its internal key records DOMAIN KEYS LIST, that no other device already has a key that would match this newly generated signature. If this fails, we can solve the DL problem: We simulate a TC using the unknown discrete logarithm of the DL instance as tsk like in the DDH reduction before. If a matching tsk is found, then we have a solution to the DL problem. Therefore, as long as solving the DL problem is computationally infeasible, Game 11 $\approx$ Game 10.

Game 12 (SP6: Unforgeability): If I_{DAA} is honest, CHECK XIII is added to $\mathcal{F}$ to prevent anyone from forging signatures by using honest DAA key tsk and its credentials. If the VC-issuer is honest, CHECK XIV prevents creating Verifiable Presentations with join credentials that were not issued by the Issuer. The unforgeability of our protocol is built on the unforgeability of the DAA-A Signature [3].

Game 13 (Revocation): CHECK XVI is added to $\mathcal{F}$. This ensures that honest devices are not being revoked. If an honest device is simulated, when a matching key in the revocation list is found, it must be the secret key of the target instance. This is again equivalent to solving the DL of the problem (Game 13 $\approx$ Game 12). □

Chapter 5

Decentralized Attribute-based Encryption

In ASSURED, as it was outlined in D4.2 [12], we leverage **Attribute-Based Encryption (ABE)** in order to provide the capability to the devices belonging to the ASSURED network to **encrypt their attestation raw data with different levels of granularity**, in order to be able to control who can access and decrypt this data **based on their attributes and privileges**, based on the **security and privacy requirements**. Here, we expand on the first version of the ASSURED ABE which was presented in D4.2, and we provide a detailed action workflow of the second version, taking into consideration the different types of attributes considered, as well as the use of Verifiable Credentials (VCs) and Verifiable Presentations (VPs) during the encoding and decoding processes.

5.1 Secure Data Management in ASSURED

The concept of ABE, which was first introduced in [19], offers a way to define asymmetric-key encryption schemes for policy enforcement based on attributes, where **both the user secret key and the ciphertext are associated with a list of attributes**. There are two types of ABE, namely **Ciphertext-Policy Attribute-Based Encryption (CP-ABE)** and **Key-Policy Attribute-Based Encryption (KP-ABE)**. In CP-ABE, a device encrypts the attestation raw data according to an access policy defined over a set of attributes, such that only the party who possesses a secret key associated with the attribute set satisfying the policy is able to decrypt the ciphertext. Conversely, in KP-ABE, ciphertexts are labeled with sets of attributes and private keys are associated with access structures that control which ciphertexts someone is able to decrypt.

Typically, ABE schemes that have been proposed in the literature operate in a **centralized** manner, that requires the presence of a trusted entity, referred to as the **Security Context Broker (SCB)**, to whom the devices send their raw attestation data. This entity not only dictates which are the attribute policies, but is also responsible for managing the encryption and decryption keys, and is responsible for encrypting and decrypting the data depending on who performs the request. However, the issue with this approach is that a very large degree of trust is placed on the SCB, since it has access to the actual data, as well as the cryptographic keys. The novelty of the ASSURED scheme in this regard, which will be presented throughout this Chapter, is that it is the **first scheme of its kind to provide decentralized ABE**, that enables encryption and decryption based on the presence and verification of specific attributes in the requesting devices. Specifically, **we enable the encryption process at the side of the data owner**, so there is no need for a central entity such as the SCB to have access to the actual data or the keys.

5.1.1 Functional Specifications in Envisioned Use Cases

As it has been previously highlighted in D4.2 [12] and throughout several deliverables, the envisioned ASSURED use cases, namely **BIBA Smart Manufacturing**, **DAEM Smart Cities**, **UTRC Smart Aerospace**, and **SPH Smart Satellites** have various security and privacy requirements pertaining to the communication and sharing services of any operational and/or attestation-related data. In Table 5.1, we outline the usage of the ABE scheme in the context of each use case towards fulfilling these requirements, particularly with regards to the ability to provide access to different sets of data with a high level of granularity.

Use Case	I want to <Action>,	so that <Reason>	**ABE Utilization**
BIBA	safeguard the data confidentiality and privacy of all attestation evidence of the loaded Real Time Location or Motion Capturing processes	only internal and external stakeholders with the appropriate attributes can have access to	All extracted control-flow and configuration traces (**attestation data**) will be encrypted using the ABE key, produced by the TPM-based Wallet, based on the attributes that need to be exhibited by the devices/stakeholders that want to process the data (based on attribute-based policies circulated by the SCB during the device registration).
DAEM	share all monitored operational data, from the deployed devices, to only those stakeholders with the appropriate privileges	confidentiality, integrity and privacy of the data is not violated	All extracted **operational data** is shared through the ASSURED Blockchain infrastructure and are safeguarded by appropriate Attribute-based Access Control mechanisms offered by the SCB. Essentially, only stakeholders that can exhibit the correct attributes through Verifiable Presentations (based on verifiable credentials that were issued and ratified during their registration and enrolment) can access the recorded data.
UTRC	encrypt all information related to the actual level of trustworthiness for each aircraft	only external stakeholders with the appropriate privileges can certify and audit the correct operation of an aircraft	All **attestation data**, prior to be shared by the SSR with the GSS, will be encrypted using the ABE Key of the SSR TPM-based Wallet. Encryption will be done based on keys that will be created by the TPM based on specific attributes that need to be exhibited by those stakeholders wishing the process the data - based on policies initially defined by the SCB during the SSR registration and enrolment process.
SPH	securely have access to the history of the state of all deployed CubeSats	their correct operation can be verified and audited for their entire lifecycle	All **attestation data** and system traces, based on which the verification process was executed, are signed using the ABE Key of the CubeSat so that when sent to the Ground Station (in bundles when such a communication is allowed) will be shared on the ledger with only those stakeholders that can exhibit the necessary attributes for accessing them.

Table 5.1: Utility of ABE in each of the envisioned ASSURED use cases.

5.2 Background & Notation

5.2.1 Preliminaries

Here, we provide some preliminaries that will be necessary in order to describe the ASSURED ABE scheme. In Table 5.2, we present the notation that will be used throughout the remainder of this deliverable in order to describe the various aspects and processes of the designed scheme.

Notation	Description
$KhEk_A$	Handle of the Endorsement Key of device A
Ek_{pub}	Public part of the Endorsement Key
$KhEk$	Handle of the endorsement key of the SCB (authority)
$KHash$	Keyed hash key
$KHash_{pub}$	Public part of the keyed hash key
$KhHash$	Handle of the keyed hash key
SH	Session handle
DCC	Command code of tpm2_duplicate
$randP$	Random 32 bytes password
$Priv$	The encrypted, by the random password and a TPM symmetric key, private part of the keyed hash key
$symSeed$	The encrypted (with the device's endorsement key) seed that created the symmetric key that encrypted the private part of the keyed hash
$sAbT$	Static attribute based on tree
rtA	Run-time attributes
Cre	The credentials needed for the initialization of the device
P	The policy that includes both the static attributes and the run-time attributes
K	A random nonce created by the TPM
$Secret$	The seed for the creation of the encryption and the HMAC key
PK	The public master attribute key
$KhEnc$	The handle of the symmetric encryption key
$HMAC_{key}$	The handle of the HMAC key
$HMAC$	The HMAC of the encrypted data
$HMACp$	The recreated HMAC of the encrypted data
Sig	The signature of the session digest
CT	CipherText
t_i	An attribute of the static attribute-based tree
P_i	The public attribute key of t_i
$shared$	The shared value that contains the static attributes that were used
D	The set of leaves of the static attribute based tree that will be used for the extraction of the secret
R	The root of the static attribute based tree
PT	Plaintext
SH	Session handle
$Nonce$	Session nonce
$SigPub$	Public part of the SCB's signing key
nP	New policy
$CpHash$	$H(nonce\|nP)$
$KhSigK$	Sey handle of the signing key of the SCB
$SigP$	Signature of the new policy

Table 5.2: Notations used in the description of the ASSURED ABE scheme.

The core elements of the ABE scheme are essentially the fundamental cryptographic functions used in order to perform encoding and decoding operations. The design, construction and instantiation of these functions will be analyzed in detail in Section 5.3, and are briefly summarized as follows:

- $HMAC(M,\ kMAC)$: A cryptographic hash function, SHA256, is used to generate the hash-based message authentication code for an encrypted message M according to the integrity key $kMAC$.
- $ENC(M, kENC)$: A symmetric encryption algorithm, which encrypts the message M with the key $kENC$.
- DEC$(C, kENC)$: A symmetric decryption algorithm, which decrypts the cipher-text C with the key $kENC$.

5.2.2 System Model

As it was previously mentioned, the ASSURED ABE scheme was designed in order to fulfill the security and privacy requirements of the data handled by the devices belonging to the supply chain ecosystems considered by ASSURED, represented by the envisioned use case demonstrators. In addition, we should be able to provide access to the encrypted data with different levels of granularity, so that only parties with specific properties can obtain access to specific sets of data, while simultaneously ensuring that no malicious or unauthorized parties are able to access such data sets. In this context, we first define the components participating in the ASSURED ABE, which essentially constitute the system model of the ABE scheme:

- **Trusted Authority**: An entity that is responsible for defining the attribute space (as it will be outlined in Section 5.2.3) and is able to generate the user attribute keys for each of these attributes. The trusted authority essentially dictates the attribute policies which must be validated in each encryption-decryption process. In ASSURED, the **Security Context Broker (SCB)** is the party acting as the trusted authority.
- **Encryptor:** The device that aims to encrypt data in a manner that makes decryption possible only by parties that possess the appropriate attributes, which are specified during encryption. The encrypted data may be *attestation data* which is generated after the execution of an attestation process and needs to be recorded to the ledger and made accessible to interested parties (mainly in the BIBA Smart Manufacturing, UTRC Smart Aerospace, and SPH Smart Satellites use cases), or *operational data* which may be extracted during the operation of devices such as smoke sensors (mainly in the DAEM Smart Cities use case).
- **Decryptor:** The device that aims to decrypt data that has been encrypted by the Encryptor using the ASSURED ABE scheme. In order to do this, the Decryptor must possess the attributes specified by the Encryptor during the encryption process. Note that the totality of these attributes has been registered during the Secure Enrollment process (which has been presented in detail in D4.5 [13]) in the form of *Verifiable Credentials (VCs)*, and in order to access a set of encrypted data, the Decryptor must create a *Verifiable Presentation (VP)* containing only the subset of attributes required in order to perform the decryption.

5.2.3 Attributes Design Space

Here, we focus on defining the sets of all possible attributes that can be considered and managed in ASSURED, and can be used in the context of ABE encryption and decryption operations. It is important to note that the ASSURED ABE scheme is agnostic to the type of attributes considered, as **the set of attributes to be used by an Encryptor is chosen by the SCB, depending**

on the type of Encryptor device and the Service Graph Chain (SGC) it operates within, and the SCB constructs the attribute tree depending on the requirements of the target SGC. Therefore, not all attributes need to be shared between all Encryptors or Decryptors in the SGC, and conversely these attributes do not need to be different between them. For example, consider the *BIBA Smart Manufacturing* use case, where the system consists of multiple manufacturing floors, and different devices (sensors, robotic arms) operate within the same manufacturing floor. In this case, all Encryptor devices operating in the the same floor will be using the same set of attributes, whereas devices in other floors will be using different attributes trees.

As aforementioned, the ASSURED ABE scheme relies on the attributes that the device possesses in order to decrypt a set of data that has been encrypted using ABE, based on the appropriate attribute-based policies. Specifically, these policies dictate **a set of attributes that need to be verified by any Encryptor/Decryptor device**, which **provide the necessary levels of assurance on the data access control**. Essentially, we do not determine the devices that can perform decryption, but rather the **attributes that devices need to exhibit that they possess in order to access a specific set of data**. These attributes require different considerations regarding the management of the policies by the underlying root-of-trust. Specifically, **short-term attributes** related to device runtime configuration and execution that might need to be updated frequently need to be managed differently than **long-term attributes**, which might remain the same throughout the operational lifecycle of the device. Consider attributes based on **hardware characteristics**, such as specifications of CPU and peripherals and characteristics that can be found in the MUD (Manufacturing Usage Description) of the device. These usually do not change, and can thus be treated differently (e.g., stored in the PCRs and NV-RAM of the Trusted Component of the device).

It should be noted that there are different types of properties to be considered, and each type should be handled differently by the ABE scheme, as it will be described in detail throughout the remainder of this Chapter. Specifically, the attributes can be categorized into **static** and **runtime**, which are defined as follows:

- **Static properties:** These cover the design space of attributes that generally do not change over time. In case they do change, this might result in a change in the state of a device, which should be protected from frequent changes. Thus, we opt to add them to the PCRs of the device TPM, so that they cannot be changed without disrupting the normal operation of the TPM, and require a device re-boot in case of a state change. Examples of such properties are *secure boot identity*, *firmware version*, and *the type of the CPU*. Some static properties may originate from the **device manufacturer** and are determined during the manufacturing process of the device, but may also include properties afterwards defined in the MUD profile of the device. For instance, the static property pertaining to the secure boot ID is one such property that is not created during the manufacturing process.

- **Runtime properties:** These capture the full spectrum of the design space, and include system properties that may change frequently during the operational lifecycle of the device, and depict the level of trust in the correctness of the operational state of a device. In D3.3 [14] we have defined the various **levels of assurance** that are captured by different types of attributes and attestation enablers. This type of properties essentially pertains to the operation landscape of the device, and differ depending on the application domain. Some such properties may depict the **current status of the encryptor/decryptor device**, which needs to be in a correct and expected state before progressing with any further operations. These properties can be attested and verified by using the various attestation enablers

offered in the ASSURED attestation toolkit, such as **Configuration Integrity Verification (CIV)** and **Control Flow Attestation (CFA)**, which are presented in detail in D3.2 [11] and D3.3 [14]. For example, such properties may be whitelists of binaries (whose correctness can be verified with CIV), or the correct execution of internal services (which can be verified with CFA).

In the context of the ASSURED ABE scheme, this distinction is made because the handling of the attributes differs between these two types. Specifically, only the static attributes are going to be converted to attribute keys, and the dynamic attributes are going to be concatenated along with the static attributes into a key restriction policy. This is because (i) the static attributes should be protected from frequent changes, and (ii) since static attributes do not change over time, these are added in the actual attribute tree that a Decryptor needs to traverse prior to being able to proceed with the creation of the Decryption keys (considering that the key restriction usage policy is also valid). Based on this categorization, Table 5.3 depicts the types of static and dynamic properties considered in the ASSURED ABE scheme, depending on the corresponding use case. Note that, in the context of the specific use cases considered in ASSURED, the static properties are common among all use cases, while the dynamic properties differ in each use case. However this is not necessary in general since, as aforementioned, the ASSURED ABE scheme is **agnostic to the type of attributes considered, as long as they can be measured**.

Property	Type	Use case	Description
Firmware Version	Static	All	The firmware version, which does not change during runtime, originates from the Manufacturer Usage Description (MUD) profile. Based on this, we can identify any vulnerabilities corresponding to this particular firmware version. If it is well-protected, we can enable devices with this firmware version to encrypt or decrypt sets of data using ABE. If it is not well-protected, we can add it to a blacklist of firmware versions (i.e., versions where several vulnerabilities have been identified) which are not allowed to participate in ABE operations.
Secure Boot ID	Static	All	When a successful secure boot operation is executed leveraging the TPM module embedded on the device, a Secure Boot ID is assigned to the device for this process, which does not change during runtime. In the context of all the envisioned use cases, we consider that a secure boot operation must be executed so that the device can be considered to be in a correct operational state, and it can be allowed to participate in ABE operations. If a secure boot process is not executed, it is not allowed to participate in ABE operations.
Device ID	Static	All	The Device ID, coming from the manufacturer, is considered in the context of the role the device plays in the overall network. Consider the DAEM use case, where different parties (e.g. police officers, fire department personnel) should have access to different types of data. Therefore, the Device ID should be tied to the type of attestation data encrypted using ABE. Therefore, if a Device ID corresponds to a police officer, it will provide different privileges with regards to encryption and decryption data compared to a public administration authority. Therefore, the Device ID (which does not change during runtime) can be used in the list of attributes which dictate which types of data can be encrypted or decrypted by the device, thus providing more fine-grained levels of granularity to the encryption and decryption process.

Whitelist of loaded binaries	Dynamic	BIBA	When a device belonging to a smart manufacturing floor (such as a sensor or a PLC) is enrolled into the framework, a list of binaries (applications) is loaded to the device. Using the Configuration Integrity Verification (CIV) enabler, we can verify whether this list is as expected. Since these devices operate within a controlled environment, there is a priori knowledge of which binaries should be loaded and launched in each sensor or PLC. If there is a deviation in this whitelist of expected binaries (e.g., an additional binary is found, or if an expected binary is missing), the device is not allowed to encrypt or decrypt data using ABE. Note that this is a dynamic property, since the binary whitelist can change during runtime.
Role tag	Dynamic	DAEM	The role tag refers to the role that corresponds to a Device ID (e.g., firefighter or police officer), so that it ban be determined what type of data this role tag can have access to. As aforementioned, this can control the level of granularity with regards to the types of data that can be accessed by different parties participating in the ASSURED framework depending on their role. Note that the role of a device may change during runtime, thus this property is considered dynamic.
Device network characterization	Dynamic	UTRC	This refers to the hash of the network interface processes running in a device belonging to an aircraft, for the communication between the aircraft and the Ground Station (GS). Depending on the type and vendor of the aircraft, this can dictate the use of a different communication medium (e.g., wireless or Ethernet) and different communication protocols. Therefore, this property refers to the list of binaries and the type of Network Interface Card (NIC) which is expected, so that a device can participate in an encryption or decryption process. This is a dynamic property, as it may change during runtime.
Pairing between CubeSat and Ground Station	Dynamic	SPH	In this use case, a CubeSat belonging to a satellite network does not broadcast, but only communicates with a particular GS, who is the handler of a specific set of satellites. Therefore, there is a strict device profiling and pairing between CubeSats and Ground Stations. In order to enable a device to participate in an encryption or decryption process using ABE, all the above types of attributes are considered (e.g., whitelist of loaded binaries and types of NICs), in order to better define this strict pairing.

Table 5.3: Static and Dynamic Properties Considered in ASSURED ABE scheme.

5.3 Towards Decentralized ABE - Conceptual Protocol Overview

Having provided all the building blocks of the ASSURED ABE scheme, as well as the entities and components participating in an encryption/decryption process, we next provide a high-level conceptual overview of the scheme. Specifically, the scheme can be broken down into three distinct main phases, namely **Initialization**, **Encryption**, and **Decryption**:

- **Initialization phase:**

  1. Consider a device that wishes to participate in the ABE scheme, either as an Encryptor or as a Decryptor. First, the device sends its static attributes (which will eventually be used as part of Verifiable Presentations) to the SCB.

2. Afterwards, the SCB creates the static attribute elliptic keys, along with a key for the device to put into its key hierarchy in order to create the encryption/decryption and HMAC keys under a key policy, containing a concatenation of the static attributes and the dynamic attributes.
3. This data is encrypted under the device's endorsement key.
4. From the side of the device, when the encrypted data is acquired, it decrypts and stores the attribute keys inside the PCRs of its embedded TPM, and puts the key provided by the SCB in the TPM's key hierarchy.

- **Encryption phase:**

1. If an Encryptor device wishes to encrypt a set of data using ABE, it first requests from its TPM to create a random nonce.
2. It uses this nonce to compute, along with the master attribute key and the key restriction policy, the encryption key seed.
3. If then creates the Encryption Key under the key provided by the SCB, bonded with the appropriate policy.
4. In order to encrypt the desired data, the device must undergo an integrity check, using both its static attributes and its dynamic attributes.
5. If the device passes the integrity check, it encrypts the desired data and creates its HMAC.
6. Then, the device computes a shared value, using the Public static attribute keys, and the aforementioned random nonce created by the TPM.
7. The device then uses its DAA key that was created during the Secure Enrollment , to perform a DAA signature operation on the key restriction policy , in order to provide a security proof that the correct key restriction policy was used during the encryption of the raw traces.
8. The device sends the encrypted data, the HMAC of the encrypted data, the signature of the policy, the public part of the signing key, its respective credential, and the shared value, to the SCB in order to be stored in the data storage engine.
9. The signature of the policy that was used in order to perform the encryption, and the public part of the signing key, are stored in the ledger.

- **Decryption phase:**

1. When a Decryptor devices wants to perform decryption on a set of ABE-encrypted data, it first retrieves the DAA-signature.
2. It uses the policy that was expected to be used in order to verify the signature provided by the Encryptor and to recreate the credential (recall the motivation provided for this verification process provided in Section 5.2.2).
3. If this verification is successful, the Decryptor uses its static attribute key to compute the encryption key seed.
4. With the extracted key seed, it creates an HMAC key in order to recompute the HMAC of the encrypted data.
5. It uses this HMAC in order to perform an authentication check, by comparing it to the one provided by the Encryptor.

6. If the authentication check is successful, the device goes through an integrity check, using both the static and dynamic attributes.
7. Upon successful completion of the integrity check, the Decryptor device decrypts the data.

In Section 5.4, we provide detailed descriptions of the aforementioned phases, as well as sequence diagrams depicting the actions followed in the context of each of these phases.

5.4 Architectural Details & Building Blocks

In Figure 5.1, we provide an overview containing all details of the ASSURED ABE scheme, along with the interactions between all involved entities. Thoughout this section, we elaborate on each of the phases of the designed protocol.

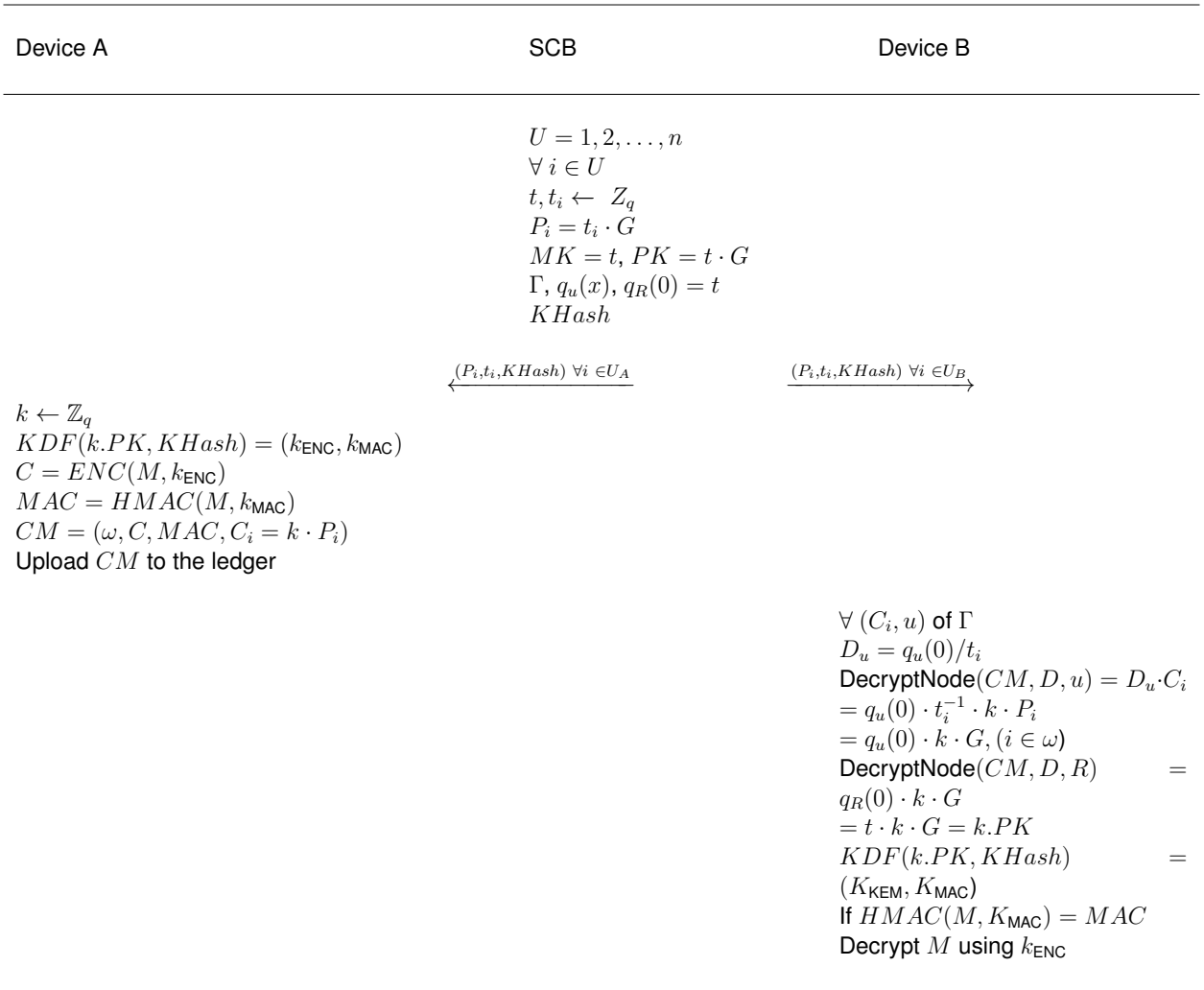

Figure 5.1: Overview of the ASSURED ABE scheme

5.4.1 Setup Phase

Next, we describe the setup interface initiated by the ASSURED Security Context Broker (SCB), performed when a device (Encryptor or Decryptor) wishes to participate in the ABE scheme. The SCB, acting as the trusted authority, is responsible for verifying the attributes by their verifiable presentations during the Secure Enrollment of the device, creating the corresponding static attribute keys for each Blockchain device, as well as setting up the access control tree. Such an access control tree would look like the one shown in Figure 5.2, where t and t_i, $i \in U$, are represented by 32-byte digests.

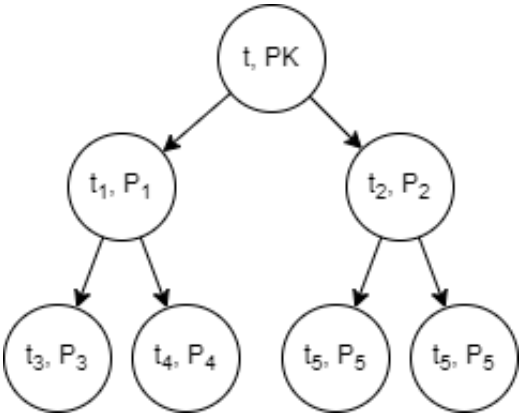

Figure 5.2: Abstract view of an access control tree created by the SCB.

Given all the above, the actions followed in the context of the Setup phase are shown in Figure 5.3, and are described as follows:

1. **SCB Setup:** A keyed hash key gets created as a child of the TPM's Endorsement key of the SCB. This key is going to be migrated to every device that gets securely enrolled to the SCB, to be used as the parent of the encryption and the HMAC key under the device's TPM's key hierarchy.

2. The attribute space in the system is defined as the universe of attributes $U = 1, 2, ..., n$. For each attribute $i \in U$, a number t_i is chosen uniformly at random from $\mathbb{Z}_q$. The public key of each attribute i is $P_i = t_i \cdot G$, where G is the group generator. The group generator is public information, known to all devices.

3. The SCB chooses t uniformly at random from $\mathbb{Z}_q$ to be the master (private) key t. Accordingly, the master public key PK is $PK = t \cdot G$. The public parameters are denoted by $Params = PK, P_1, ..., P_U$.

4. The SCB defines an access tree Γ by creating a public polynomial $q_u(x)$ with order of $d_u - 1$ should be defined for each node u in the access tree Γ in top-down manner such that $q_u(0) = q_{parent}(index(u))$, where d_u is the threshold of the node u. For the root R of the access tree Γ, set $q_R(0) = t$.

   In order to prevent the CPA attack presented in [21], the creation of q_u at each node, should not only depend on the index of the node u in the access tree, but also on an extra n-bit randomness $r_{\mathbb{A}}$ that depends on the access policy defined at this node. Thus, q_u at the node u is defined by $q_u(PRF(r_{\mathbb{A}}, index_u))$ for some pseudo-random function $PRF : \{0,1\}^n \times \{0,1\}^* \rightarrow \mathbb{Z}_q^*$ that takes a random $r_{\mathbb{A}}$ and the index of the node where the polynomial is defined.

5. **Device Setup:** We use the device DAA Key created during the Secure Enrollment for the encryption of the data that is not public and that the SCB wants to securely send to the device.

6. The Device sends to the SCB the public part of the Endorsement Key of its TPM, along with the name of the NAME of the DAA Key.

7. The SCB generates a random password, which is used to double encrypt the private part of the keyed hash, as an extra level of security to provide freshness every time a device challenges the SCB to duplicate the keyed hash key. The SCB creates a random seed that afterwards creates a symmetric key, which is going to encrypt the private part of the key. Then, the TPM encrypts the random seed with the Endorsement Key of the device that initiated the challenge. Finally, with the random password generated by the SCB, the TPM encrypts the encrypted private part of the keyed hash.

8. The SCB encrypts the static attribute-based keys and the random password for each user under the device's TPM Endorsement Key, and wraps them with the NAME of the DAA Key using the command TPM2_MakeCredential.

9. The SCB generates a **Key Policy**, which is the concatenation of the hashed static attributes and run-time attributes, along with the command codes of the commands that need to be executed. This Key Policy is bonded with the Encryption/Decryption key, in order to check the integrity of the device.

10. The SCB sends the respective Key Policy to the device, containing a concatenation of both the static and the run-time attributes, as well as the encrypted static attribute-based tree, in order for the device to be able to compute the random key seed for the encryption key and the HMAC key and to compute the shared value from which the Decryptor is going to extract the random key seed, along with the random password, and the double encrypted private part of the keyed hash and the encrypted (by the Device's Endorsement key) seed that is used to generate the inner password and the public part of the keyed hash.

11. The device decrypts the encrypted static attribute-based tree, which is afterwards stored inside the PCRs of the TPM, and the random password generated by the SCB using TPM2_ActivateCredential.

12. The device using the decrypted random password, the public part of the keyed hash, the encrypt seed of the inner encryption key and the double encrypted private part of the keyed hash safely imports the SCB keyed hash as a child of its Endorsement key using TPM2_Import.The keyed hash is imported to every device in order to authorize the creation of derived TPM keys.

5.4.2 Encrypt Phase

In order to describe the steps that must be taken in order to encrypt a clear message, we will assume that an Encryptor device (referred to as Device A in the following) wants to encrypt a message to be uploaded on the ASSURED ledger. The steps that an Encryptor device must complete are given in Figure 5.4 and are described as follows:

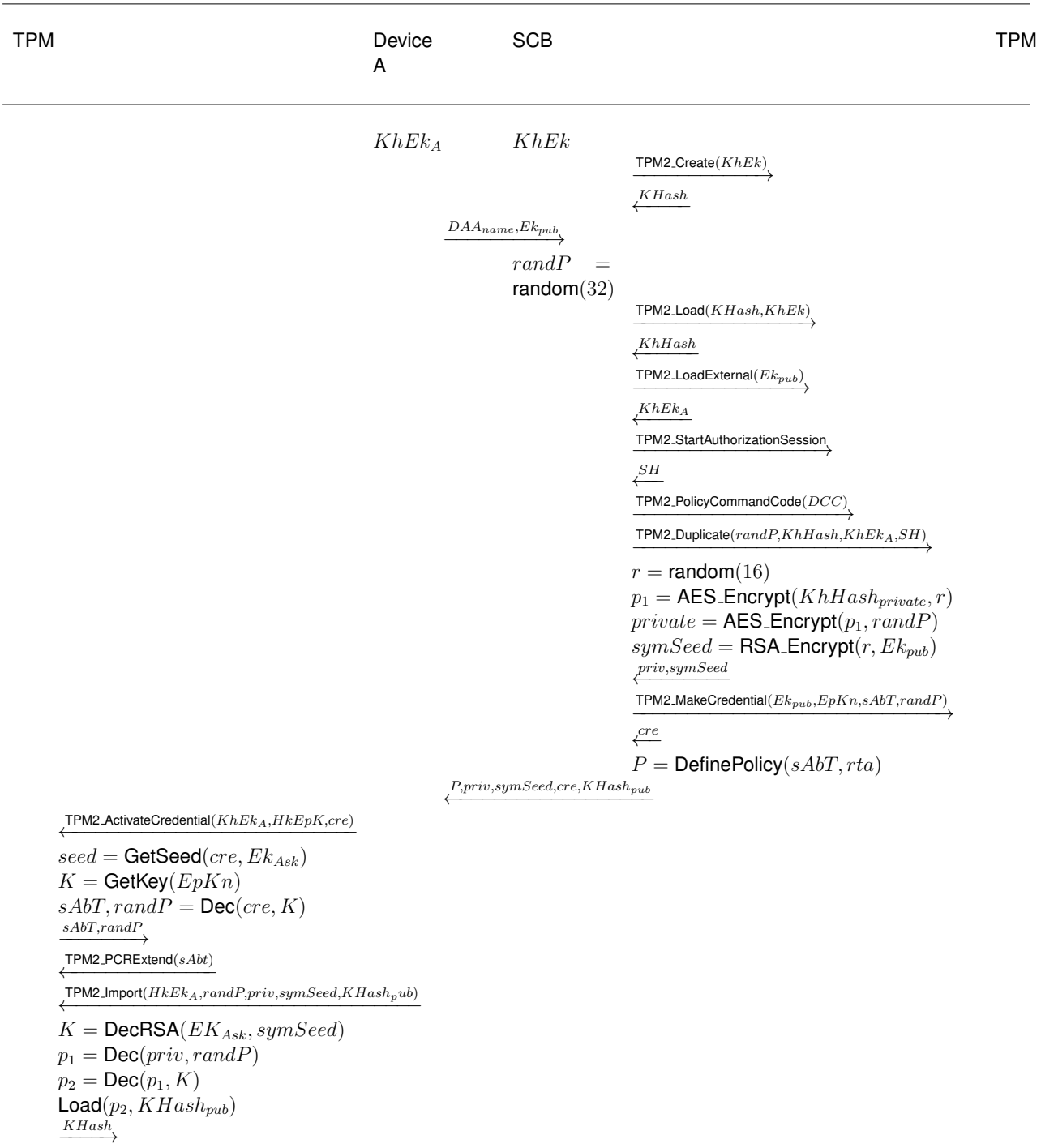

Figure 5.3: ASSURED ABE Setup Phase Sequence Diagram Using TPM 2.0 Commands

1. The Encryptor Device A must create a random secret k generated by TPM2_GetRandom, and performs the scalar multiplication $k * PK$.

2. Device A takes the secret value $k * PK$, which is going to be used as a co-seed along with the private part of the imported keyed hash key, in order to create a symmetric encryption key bonded with the key restriction policy that the SCB has created during the initialization phase and an HMAC key, as children of the Keyed hash key. This is achieved by executing

the TPM command TPM2_CreateLoaded.

3. Device A executes TPM2_policyPCR to load the static attributes that are stored inside the PCRs of the TPM to a session. It also executes policyOR with the run-time attributes as an argument in order to check the key restriction policy, that the SCB has created during the initialization of the protocol, against the policy that is created from the static attributes and run-time attributes taken as input.

4. Device A, with the newly created Encryption Key, will encrypt the data by executing the command TPM2_EncryptDecrypt2.

5. The Device A uses the DAA key to perform a DAA signature for the audited session digest, to use this signature as a proof of the correct policy usage.

6. With the newly created HMAC key, Device A executes TPM2_HMAC to compute the digest of the encrypted data.

7. Device A reads the static attributes from the corresponding PCRs, and computes the shared value $C_i = kP_i$ for $i \in \omega$, where $\omega \subset U$ is a set of required attributes for Device A.

8. Device A uploads the encrypted data, the HMAC of the encrypted data, and the shared value C_i.

5.4.3 Decrypt Phase

Any Decryptor device (henceforth referred to as Device B) with matching attributes can successfully complete the decryption of device A uploaded on the ledger. The steps followed in order to perform a decryption process are depicted in Figure 5.5, and are described as follows:

1. Device B, that has the correct policy that had to be used during the encryption of the data, verifies the DAA signature that Device A provided. If the signature verification is successful, it is proven that Device A encrypted the data under the correct policy, thus Device B can continue the decryption algorithm.

2. For each C_i on the nodes u of Γ, Device B calculates the corresponding decryption node key denoted by D: $D_u = q_u(0)/t_i$. Here, $i = attr(u)$ and t_i is the attribute key assigned to users from $\mathbb{Z}_q$ in the Setup phase by the SCB, and t_i^{-1} is the inverse element of t_i over finite field $\mathbb{Z}_q$.

3. Device B runs the decryption algorithm $DecryptNode(C_i, D, u)$ for a node u in the access tree, which is defined as an recursive procedure. For a leaf node u, where attributes are clearly defined, let $i = attr(u)$. an For this node, the key is computed as $DecryptNode(C_i, D, u) = D_u * C_i = q_u(0) \cdot t_i^{-1} \cdot k \cdot P_i$. It should be stated that the output of $DecryptNode(C_i, D, u)$ is an element in an elliptic curve group or $\perp$. Accordingly, for the root node R of the access tree, the key is computed as $DecryptNode(C_i, D, R) = q_R(0) \cdot k \cdot G = t \cdot k \cdot G = k \cdot PK$.

4. Device B, with the extracted secret value $k * PK$, executes TPM2_CreateLoaded using $k * PK$ as a co-seed, along with the private part of the imported from the SCB keyed hash, to create the decryption key bounded by the policy provided by the SCB and the HMAC Key, as children of the keyed hash key.

Figure 5.4: ASSURED ABE Encrypt Phase Sequence Diagram using TPM2.0 Commands

5. The device B, using the HMAC key, computes the digest of the encrypted data and compares it to the one provided by the Encryptor.

6. If the authentication is successful, Device B executes TPM2_policyPCR to load the static attributes that are stored inside the PCRs of the TPM to a session. Then, it performs a policyOR operation with the run-time attributes as an argument in order to check the key restriction policy, that the SCB has created during the initialization of the protocol, against

the policy that is created from the static attributes and the run-time attributes taken as input.

7. If the integrity check is successful, Device B executes TPM2_EncryptDecrypt2 to Decrypt the encrypted message using the recreated decryption key.

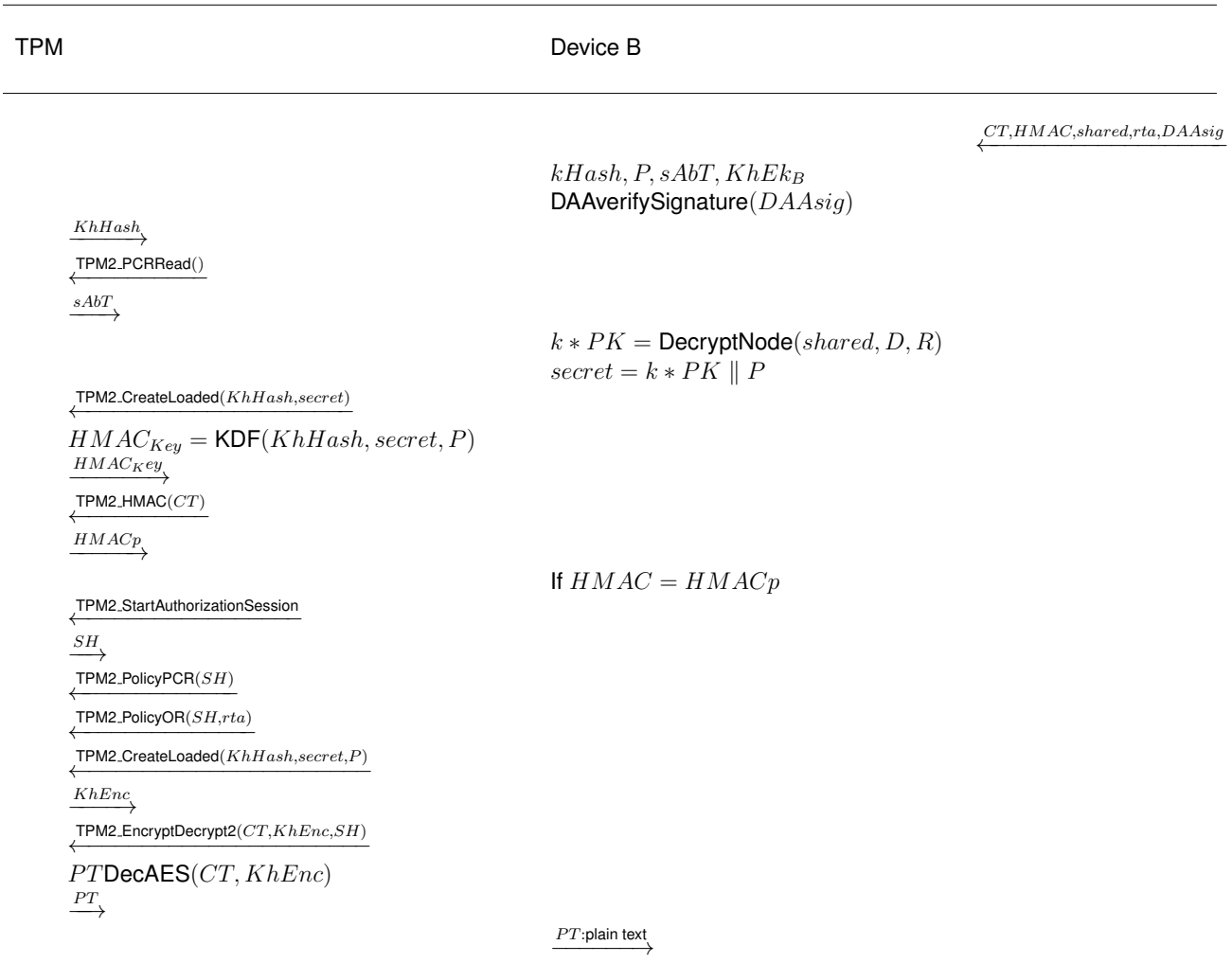

Figure 5.5: ASSURED ABE Decrypt Phase Sequence Diagram using TPM2.0 Commands

5.4.4 Attribute List Update

Any Device that is enrolled in the network can update both its static attributes and its run-time attributes. In case a device wants to update its static attributes, it will first have to reboot its system and afterward be securely re-enrolled to the network. However, in case the device has to update its dynamic run-time attributes, the following steps are followed, which are also depicted in Figure 5.6:

1. In order to initiate the update process for the dynamic attribute list, the SCB requests from the device a session nonce.

2. With this session nonce, the SCB signs the new policy that contains the concatenation of the run-time and the static attributes.

3. The SCB sends the new signed policy to the corresponding device, along with the corresponding $cpHash$.
4. The device then verifies the signature, using the session that provided the nonce sent to the SCB.
5. If the verification is successful, the old policy is replaced with the new one.

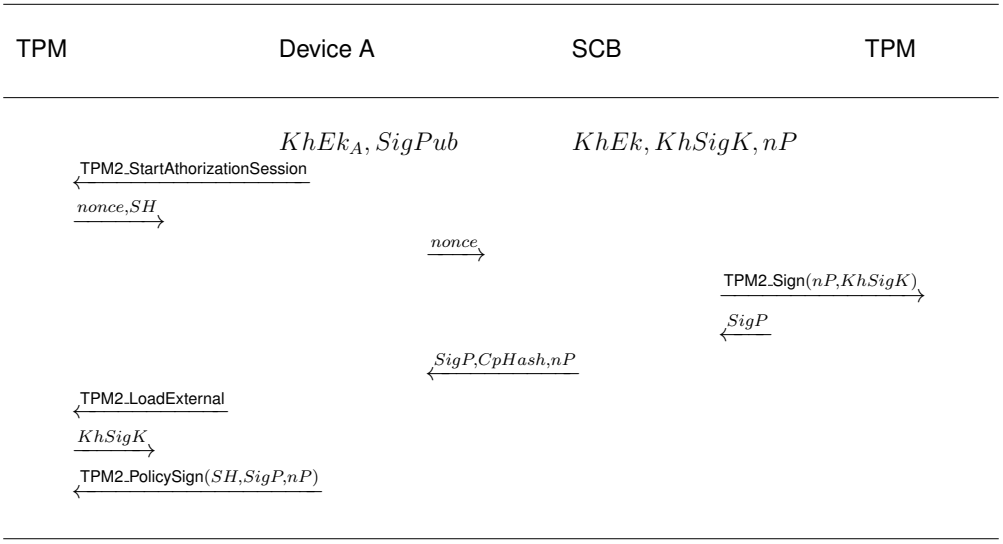

Figure 5.6: ASSURED ABE Key Policy Update Sequence Diagram

5.5 ASSURED ABE Security Analysis

5.5.1 ABE Security Model

The Selective-Set security model is used to prove the security of an ABE scheme, in which the two messages encrypted by a KP-ABE are indistinguishable under chosen plaintext and attribute-set attack. The attribute-based Selective-Set model is based on a game, which is played by a challenger and an adversary.
The security notion of ciphertext IND-CPA in the selective set model [24] is defined as the game played between a challenger $\mathcal{B}$ and an adversary $\mathcal{A}$ as follows:

- Initialization: The adversary declares the attributes set γ that he wants to attack:
- Setup: The challenger runs the Setup algorithm in ABE scheme and sends the public key parameters to the adversary.
- Phase 1: The adversary is permitted to make many queries for the decryption keys for many access structures $\mathbb{A}_j$, where $\mathbb{A}_j(\gamma) = 0$, for all j.
- Challenge: The adversary submits two equal-length messages M_0 and M_1 to challenger. The challenger flips a random coin v and encrypts M_v under the attributes set γ. Thereafter, the ciphertext is sent to the adversary.

- Phase 2: Repeat phase 1.
- Guess: The adversary outputs a guess v' of v. In the game, the advantage of the adversary is defined as

$$\varepsilon = Pr[v' = v] - 1/2$$

A KP-ABE scheme is said to be IND-CPA secure in the selective-set model if an adversary can win the game in polynomial time with at most a negligible advantage.

Note that in the weaker security notion of OWE-CPA [20] in the selective-set model, the adversary $\mathcal{A}$ is given a ciphertext CM during the Challenge phase. A wins the game if the correct plaintext is recovered during the Guess phase such that $M = Decrypt(CM)$.

5.5.2 ABE Security Proof

Since we have defined the security model of the ABE protocol, we can now proceed with the security proof of the proposed scheme, leveraging the Discrete Logarithm (DL) and Decisional Diffie-Hellman (DDH) assumptions. This can be expressed as follows:

Definition 1. *(The Discrete Logarithm (DL) assumption [18]) Given $y \in G$, find an integer x such that $g^x = y$.*

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

Based on the above definitions the security of the proposed ABE scheme is proved in the attribute-based Selective-Set model. Similar to the existing ABE schemes, the method of reduction to absurdity is used to prove the scheme's security. Since the security of our scheme depends on the Elliptic Curve Decisional Diffie–Hellman problem, reducing the hardness of the ECDDH assumption is made.

Theorem: Suppose that an adversary $\mathcal{A}$ can attack our scheme successfully in the attribute-based selective-set model in polynomial-time, a simulator $\mathcal{B}$ can solve the ECDDH problem with non-negligible probability.

Proof: Firstly, the challenger sets the elliptic curve group G_E with the order of q over $\mathbb{F}_p$ and generator G. Then, the challenger flips a fair binary coin μ and randomly chooses a, b, z from $\mathbb{Z}_q^*$. If $\mu = 0$, it sets $(A, B, Z) = (cG, dG, cdG)$; otherwise, it sets $(A, B, Z) = (cG, dG, zG)$. It is assumed that the universe of attributes U is defined.

- **Initialization**: The simulator $\mathcal{B}$ get the attributes set γ, which is the adversary $\mathcal{A}$ wishing to attack. It is assumed that the access tree corresponding to the attributes set γ is Γ.
- **Setup**: The simulator $\mathcal{B}$ runs the Setup algorithm of the proposed scheme to set the system's public key parameters and sends them to the adversary $\mathcal{A}$.
  - $\mathcal{B}$ samples $KHash \leftarrow \mathbb{Z}_q$
  - $\mathcal{B}$ sets the system parameter $Y = A = cG$
  - $\mathcal{B}$ sets Y_i according the following principles $\forall i \in U$

1. If $i \in \gamma$, $\mathcal{B}$ randomly chooses $r_i \leftarrow \mathbb{Z}_q^*$ and sets $Y_i = r_i G$ and $y_i = r_i$
2. If $i \in U - \gamma$, $\mathcal{B}$ randomly chooses $\beta_i \leftarrow \mathbb{Z}_q^*$, and sets $Y_i = \beta_i B = d\beta_i G$ and $y_i = d\beta_i$

– $\mathcal{B}$ sends the system public key parameters $\{Y, Y_i, i \in U\}$ to $\mathcal{A}$. It is obvious that Y, Y_i corresponds to $\{PK, P_i\}$ of the proposed ABE scheme.

- **Phase 1**: The adversary $\mathcal{A}$ can make many queries for the decryption key corresponding to any access structure Γ^*, where the challenge set γ does not satisfy Γ^*, that is $\Gamma^*(\gamma) = 0$. In order to make the adversary $\mathcal{A}$ can reconstruct the decrytion key for the access structure Γ^*, the simulator $\mathcal{B}$ needs to assign a polynomial Q_u with a degree of d_u to every node u in the access tree Γ^* . The polynomial Q_u for each node u in Γ^* is defined as $Q_u(x)$ to be $q_u(x)$ in the proposed scheme. The shared secret is set to be the constant of the root's polynomial, that is $Q_R(0) = c$. The secret key corresponding to each leaf node x in Γ^* is given by its polynomial as follows:

  $D_x = Q_x(0)/r_i$=

  – $q_x(0)/r_i$, if $(att(x) \in \gamma)$
  – $q_x(0)/\beta_i d$, if $(att(x) \notin \gamma)$

  where $i = attr(x)$.

  The set $(D_x, x \in \Gamma^*)$ is the secret key for the access structure Γ^* , which is sent to the adversary $\mathcal{A}$.

- **Challenge**: The adversary $\mathcal{A}$ will submit two challenge messages M_0 and M_1 to the simulator $\mathcal{B}$. $\mathcal{B}$ flips a fair binary coin τ, and computes the ciphertext of M_τ. In order to encrypt the message M_τ, $\mathcal{B}$ randomly chooses k from $\mathbb{Z}_q^*$ and computes the encryption and the authentication keys as the output of $PRF(kY, KHash) = (K_x, K_y)$, if K_x or K_y is 0, then resample k. Set $\texttt{SHARED} = kY$, $C_i = r_i B,\ i \in \gamma$. $\mathcal{B}$ calculates $C = ENC(M_\tau, K_x)$ and $MAC = HMAC(M_\tau, K_y)$, and transmits the cipher-text $(\gamma, C, MAC, C_i,\ i \in \gamma)$ to $\mathcal{A}$.

  1. If $\tau = 0$, then $Z = cdG$. If k is set to be d, there should be $\texttt{SHARED} = kY = dcG = Z$, and $C_i = kY_i = dY_i = dr_i G = r_i B$, where, $i \in \gamma$.
  2. If $\tau = 1$, then $Z = zG$. If k is set to be z, it turns out that $\texttt{SHARED} = zG$.

  Since c, d and z are random numbers, SHARED will be a random element of G_E from the adversary $\mathcal{A}$'s view and it cannot obtain any sensitive information about M_τ from the cipher-text.

- **Phase 2**: Both the simulator and the adversary perform exactly as they did in Phase 1.

- Guess: The adversary $\mathcal{A}$ sends a guess τ' of τ to $\mathcal{B}$.

  1. If $\tau' = \tau$, the simulator $\mathcal{B}$ outputs $\mu' = 0$ to indicate that it was given a valid ECDDH-triple (A, B, Z).
  2. If $\tau' \neq \tau$, the simulator outputs $\mu' = 1$ to indicate that it was given a random triple (A, B, Z).

Note that the probability that μ' matches the value of μ that is set in the initialization phase is half as this is a random guess from the simulator.

$Pr[\tau \neq \tau'] = Pr[\tau = \tau'] = 1/2$ when $\mu = 1$ since this a random guess if $\mathcal{B}$ was given random triple (A, B, Z).

In our ECDDH game,

$$Pr[\tau' = \tau] = 1/2Pr[\mu' = \mu = 0]+1/2Pr[\mu' = \mu = 1] = 1/2(1/2+\varepsilon)+(1/2)(1/2) = 1/2+\varepsilon/2$$

It turns out that the overall advantage of the simulator $\mathcal{B}$ in the ECDDH game is

$$(Pr[\tau' = \tau] - 1/2 = 1/2 + \varepsilon/2 - 1/2 = \varepsilon/2)$$

which conflicts with the fact of harness of the ECDDH problem for non-negligable ε. This completes the proof.

Chapter 6

Performance Analysis & Evaluation

6.1 ASSURED TC-enabled Verifiable Credentials for Selective Disclosure

6.1.1 Experimental Setup & Implementation

The proposed protocols have been implemented using the C programming language. For generic cryptographic operations, OpenSSL 1.1.1 was used, and for Pairing-Based Cryptography (PBC) the Apache Milagro Crypto Library (AMCL) was used. To access the TPM, we used IBM's TPM Software Stack (IBMTSS) version 1.6. The software was also tested to run on Android devices by exporting the functionality to libraries with interfaces to the individual resources. As Android devices currently do not have a TPM as a TC, the performance evaluation was performed on a laptop. The laptop was equipped with an Infineon SLB9670 TPM 2.0 (Rev. 1.38) and an Intel i7-8665U 1.90–2.11 GHz CPU. For the experiments, the issuers were running locally on the same machine, but the structures were marshaled and unmarshalled, giving us the closest timings without accounting for network delays. All experiments were executed 50 times to acquire a realistic mean value with a 95% Confidence Interval, using 10 attributes where 5 were disclosed. The timings were measured using *real time* and summarized to give an idea of the total time consumption. Note that these summarizations, due to the nature of the timings, will be notably higher than the overall execution time.

Activity	Mean	± (95% CI)
Issuer Create VC	32.29 ms	1.00 ms
Holder Verify VC	8.95 ms	0.27 ms
Total Accumulated Time	41.24 ms	1.27 ms

Table 6.1: Issue Verifiable Credential

6.1.2 Performance Evaluation

We present the performance analysis and evaluation of the Verifiable Credentials for Selective Disclosure scheme discussed in Chapter 4. We implemented our scheme and timed each component using real-time measurements for both the TC as well as the Holder device since they are equipped with different processing units as detailed in Section 6.1.1. Four experiments were carried out, including 1) Initialization of the Policy Index, which corresponds to the creation, authorization, protection, and storage of a key restriction policy; 2) The issuance of a DAA credential (DAA Join), creating the DAA Key, certifying it, and verifying the policy and PI; 3) The Issuance of

a Verifiable Credential; and 4) The creation and verification of a VP, which checks the satisfaction of the key restriction usage policies and the generation of attribute attestations.

Activity	Mean	± (95% CI)
Host Calculations	33.63 ms	4.04 ms
Total TPM Time	1324.36 ms	44.80 ms
TPM2_StartAuthSession	52.48 ms	2.67 ms
TPM2_PolicyCommandCode	1.51 ms	0.03 ms
TPM2_PolicyOR	3.11 ms	0.09 ms
TPM2_PolicyAuthorizeNV	326.28 ms	7.41 ms
TPM2_Commit	176.63 ms	3.23 ms
TPM2_Hash	119.27 ms	9.41 ms
TPM2_StartAuthSession	51.32 ms	2.64 ms
TPM2_PolicyPCR	2.57 ms	0.07 ms
TPM2_PolicySigned	141.02 ms	2.06 ms
TPM2_PolicyOR	3.18 ms	0.10 ms
TPM2_PolicyAuthorizeNV	325.37 ms	7.74 ms
TPM2_Sign	121.62 ms	9.35 ms
Total Create Time	1357.99 ms	48.84 ms
Verify Presentation	85.40 ms	5.02 ms

Table 6.2: Create & Verify Verifiable Presentation

Table 6.1 shows the timings for the issuance of a VC. It is evident that this operation does not invoke significant overhead for the Issuer. Within 42 ms, without accounting for network and other delays, a device-bound VC can be issued to and verified by a Holder. This low-latency issuance is especially significant, as a large set of credentials potentially has to be created on a daily basis by an Issuer, thus, showcasing the scalability of our protocol. Table 6.2 depicts the timings for creating and verifying a VP. 91% of the time consumed by the TPM is directly related to satisfying the key restriction policies for enforcing the requirements of Device Binding and unforgeability. Only around 120 ms is directly related to producing a signature from a hardware-based key. The remaining of the operations take only 34 ms and include the selective disclosure of attributes. As this is not related to the TPM, this shows that DOOR can effectively handle selective disclosure and that the primary overhead is related to hardware-based security. On the same note, verifying the presentation is very effective as operations do not require a TPM, but instead rely on trust in the Issuers and only require the two public keys. The last two experiments are related to phases that are not expected to occur often. These include the initialization and provision of the policy index and the creation and verification of the DAA Key as shown in Table 6.3 and 6.4 respectively. Creating the PI is fairly fast, requiring only around 125ms for I_{DAA} and around 460ms for the TPM operations. However, this is an important step, for reducing the occurrence of re-issuing DAA keys due to policy updates. Creating and issuing a new DAA Key takes around 3.5 seconds, which is primarily due to proving correct constructions.

Activity	Mean	± (95% CI)
Issuer Create/Sign Policy	125.27 ms	4.70 ms
Total TPM Time	459.9 ms	14.74 ms
TPM2_DefineSpace	108.61 ms	2.59 ms
TPM2_LoadExternal	26.03 ms	2.65 ms

TPM2_NV_ReadPublic	41.43 ms	1.45 ms
TPM2_StartAuthSession	52.53 ms	2.66 ms
TPM2_PolicySigned	139.54 ms	2.06 ms
TPM2_NV_Write	91.75 ms	3.33 ms
Total Calculated Time	585.17 ms	19.44 ms

Table 6.3: Create a policy index and authorizing/writing a policy to it

Activity	Mean	± (95% CI)
Issuer Calculations	10.56 ms	1.92 ms
Host Calculations	37.20 ms	2.98 ms
Total TPM Time	3510.05 ms	69.03 ms
TPM2_CreatePrimary	282.68 ms	2.18 ms
TPM2_ActivateCredential	367.50 ms	2.17 ms
TPM2_StartAuthSession	52.79 ms	2.66 ms
TPM2_PolicyCommandCode	1.59 ms	0.18 ms
TPM2_PolicyOR	3.11 ms	0.10 ms
TPM2_PolicyAuthorizeNV	324.19 ms	7.72 ms
TPM2_Commit	174.09 ms	2.40 ms
TPM2_Hash	104.76 ms	2.45 ms
TPM2_StartAuthSession	52.11 ms	2.63 ms
TPM2_PolicyPCR	2.57 ms	0.06 ms
TPM2_PolicySigned	139.67 ms	2.05 ms
TPM2_PolicyOR	3.09 ms	0.10 ms
TPM2_PolicyAuthorizeNV	326.41 ms	6.93 ms
TPM2_Sign	110.19 ms	2.72 ms
TPM2_StartAuthSession	51.84 ms	2.59 ms
TPM2_PolicyCommandCode	1.52 ms	0.02 ms
TPM2_PolicyOR	3.18 ms	0.09 ms
TPM2_PolicyAuthorizeNV	324.82 ms	7.51 ms
TPM2_CertifyNV	725.55 ms	21.32 ms
TPM2_ActivateCredential	384.15 ms	3.15 ms
Total Calculated Time	3557.81 ms	73.93 ms

Table 6.4: Create a DAA Key and acquire a DAA credential

6.2 Attribute-based Encryption

In Section 5.4, we provided detailed descriptions for the four phases comprising the ASSURED ABE scheme. Here, we provide experimental results for each of the TPM commands executed as part of each of the four phases. The results for the **Setup Phase** are given in Table 6.5, for the **Encryption Phase** in Table 6.6, for the **Decryption Phase** in Table 6.7, and for the **Update Phase** in Table 6.8. Note that a SW-based TPM was used in order to extract the provided results.

Command	Minimum [s]	Max [s]	Average [s]
Start_up	0.000479143	0.001296833	0.0006908309
Create_Primary1	0.115632991	0.208802985	0.1505438672
Create_Primary2	0.010191826	0.100167967	0.0520305772
FlushContext	0.087554294	0.09166006	0.0884638071
Import	0.045604294	0.049391762	0.046555534
Activate_Credential	0.08468566	0.090886026	0.0860714648
PCR_Extend	0.087758074	0.091913492	0.0883325196
Initialization	0.220555491	0.228811804	0.22297459

Table 6.5: RPI results - Setup phase

In Table 6.5, we provide timings for each individual command comprising the **Setup phase** of the ABE scheme, whose action workflow was presented in detail in Section 5.4.1. The process begins with the Start_up command, which is responsible for performing startup on the TPM of a device that wishes to participate in an ABE operation. Then, the Create_Primary1 command creates the **Endorsement Key (EK)** of the TPM. Note that this is only necessary in the case of a SW-based TPM, as an EK is already embedded in HW-based TPMs, so this creation command is not needed. The Create_Primary2 command creates the **DAA Key**, which will be used in order to wrap attributes and keys used in the context of the ABE protocol. The Import command receives the SCB keyed hash as a child of its EK in order to authorize the creation of derived TPM keys. Activate_Credential decrypts the encrypted static attribute-based tree received from the SCB, which is afterwards stored in the PCRs of the TPM and the random password generated by the SCB using the PCR_Extend command.

In these results, the Import command contains a variety of operations, such as RSA decryption, AES encryption, and insertion of a new key (encrypted with the device EK) in the device hierarchy. While intuitively we would expect this command to be quite computationally intensive, we observe that the time required for its execution is comparable to the rest of the commands of the process. This can be attributed to the fact that the RSA and AES encryption and algorithms are very well optimized for both SW- and HW-based solutions, thus resulting in lower execution times. Finally, note that, in this and all subsequent results, Flush_Context and Load are used as auxiliary implementation commands, and their timings may vary depending on the architectural details of the target application and the type of TPM used.

Command	Minimum [s]	Max [s]	Average [s]
Load1	0.041668282	0.097915295	0.086626373
GetRandom	0.087272826	0.096445464	0.089128189
PCRread1	0.084913646	0.090138905	0.086869641
CreateLoaded1	0.084786446	0.099816885	0.089532843
StartAuthSession1	0.087639353	0.095790365	0.088750047
GetRandom	0.087272826	0.096445464	0.089128189
PolicyPCR1	0.087390458	0.095857822	0.088814546
PolicyOR1	0.087176325	0.096021267	0.089060932

Enc	0.087818471	0.092457825	0.089202061
Flush_Context1	0.087526111	0.092044875	0.088896517
Flush_Context2	0.08772749	0.096141917	0.088997436
ReadPub	0.087694267	0.09881858	0.0900626889
commit	0.084670112	0.092338501	0.08761665077
hash	0.08772174	0.09579848	0.089413322
sign	0.087703755	0.092146497	0.089556716
CreateLoaded2	0.087836438	0.095818798	0.0897528454
HMAC1	0.087308779	0.092237415	0.088549177
Flush_Context3	0.08789345	0.092359462	0.0893586569
Flush_Context4	0.087685309	0.096126202	0.089384415
Encryption	1.657915699	1.73186465	1.7108687586

Table 6.6: RPI EVAL (SW TPM) Encrytpion

In Table 6.6, we provide timing results for the **Encryption phase** of the ABE scheme, whose action workflow was presented in detail in Section 5.4.2. Here, the GetRandom command begins the encryption phase, by generating a random number that will be used as a seed for the creation of the symmetric attribute keys, and PRCread1 loads the shared secret from the PCR bunkers of the TPM. Using these two, the CreateLoaded1 command creates the symmetric key. Next, StartAuthSession1 starts an authentication session and PolicyPCR1 loads the static attributes from the corresponding PCR into the session. Then, PolicyOR1 loads the runtime attributes extracted by the Tracer into the session in order to check the key restriction usage policy. The ABE encryption is performed by the Enc command, by using the **AES-128** algorithm. Afterwards, the key policy digest is pseudonym signed with the sign command. Then, CreateLoaded2 creates the HMAC key and HMAC1 calculates the **SHA-256** hash of the encrypted data under the created key.

In these results, we observe that the encryption and HMAC commands do not induce any significant overhead compared to the rest of the commands of the process. This is because, as aforementioned, the implementations for both the AES and SHA algorithms are quite efficient, and enable the fast execution of these operations.

Command	Minimum [s]	Max [s]	Average [s]
PCRread2	0.040956647	0.046038826	0.0425950741
Load2	0.085529309	0.096094357	0.0897663086
CreateLoaded3	0.087641028	0.096423023	0.0892710421
HMAC2	0.087273588	0.095624951	0.089328196
Flush_Context5	0.087781278	0.096349862	0.09070446389
StartAuthSession2	0.087745955	0.096259264	0.089128189
PolicyPCR2	0.087516736	0.095694547	0.08961389142
PolicyOR2	0.08730007	0.095979596	0.08881217314
CreateLoaded4	0.08790548	0.096296282	0.0900618131
Decrypt	0.087694665	0.099167148	0.08914648377
Flush_Context6	0.08762797	0.09584955	0.088933806
Flush_Context7	0.087631657	0.096054479	0.08926993656
Flush_Context8	0.087631532	0.099420784	0.0902540108
Decryption	1.193754507	1.241716477	1.21105765505

Table 6.7: RPI EVAL (SW TPM) Decryption

In Table 6.7, we provide timing results for the **Decryption phase** of the ABE scheme, whose action workflow was presented in detail in Section 5.4.3. Here, when a device aims to decrypt a set

of ABE-encrypted data, it uses the PCRread2 command to load the shared secret, and the Load2 command to load the policy digest signed with the pseudonym as mentioned in the Encryption phase. , and the VerifySignature verifies the signature of the expected policy, in order to ensure that the encryptor device used the correct policy to perform the encryption. CreateLoaded3 creates a new HMAC key using **SHA-256**, as well as the decryption key binded by the policy provided by the SCB using **AES-128**. This is compared with the HMAC key of the encryptor, and if they are identical, the decryption process continues. Note that this comparison is near-instantaneous, and is therefore not included in the steps presented in the Table. Then, StartAuthSession2 starts a new session, PolicyPCR loads the static attributes stored in the PCRs of the decryptor device's TPM into the session, and PolicyOR checks the key restriction usage policy using the runtime arguments as an argument. The corresponding symmetric encryption key is created with CreateLoaded4, and the decryption of the ABE-encrypted data is performed with Decrypt. This concludes the decryption operation.

As in the encryption case, we observe that the HMAC and decryption processes are computationally efficient compared to the rest of the decryption process, due to the efficiency of the implementation of the decryption algorithms for SHA-256 and AES-128, respectively.

Command	Minimum [s]	Max [s]	Average [s]
StartAuthSession	0.04173954	0.096301811	0.08843659872413796
LoadExternal	0.040983091	0.048806034	0.043729860402298856
PolicySigned	0.087165033	0.095862643	0.09028022372413791
FlushContext1	0.087844649	0.096133994	0.09023340357471264
FlushContext2	0.087756525	0.145211128	0.09166713181609193
Update	0.579007689	0.693202488	0.6336534370804595

Table 6.8: RPI EVAL (SW TPM) Update

Finally, in Table 6.8, we provide results for the process followed when an **Attribute List Update** process is performed, when a device aims to update its static attributes and/or runtime attributes. In this case, after the device is rebooted and securely re-enrolled to the network, StartAuthSession creates a session nonce to ensure freshness and avoid replay attacks, and the SCB signs the new policy containing the updated attributes using this session nonce. Then, this policy is loaded to the device TPM with the LoadExternal command, and PolicySigned checks if the policy has been signed under the correct nonce. If the verification is successful, the policy is stored in the device TPM and the update process is concluded.

Chapter 7

Conclusions

In this deliverable, we provided a detailed description of two core components of the ASSURED framework. Namely, (i) the **TPM-Based Wallet**, which enables devices to interact with the ASSURED Blockchain infrastructure in a manner that utilizes the TPM as a hardware-based root-of-trust to provide the **Level of Assurance (LoA)** guarantees required by various services in a privacy-preserving manner when utilizing the ASSURED data management protocols and crypto primitives, and (ii) the **Attribute-Based Encryption (ABE)** scheme, which utilizes the TPM-based Wallet in order to provide devices with the capability to encrypt data, such as raw traces used as attestation evidence, in a manner that only enables users that can demonstrate the required attributes through a VP to decrypt the data.

The final version of the ASSURED TPM-based Wallet operates within the **Self-Sovereign Identities (SSI)** concept, which aims to shift the notion of trust to the devices themselves, and provides devices with the capability to make verifiable claims regarding their identity and the correctness of their operational state. We achieve this through the issuance of **Verifiable Credentials (VCs)** that contain the entire set of device attributes that can be used in this regard, and the creation of **Verifiable Presentations (VPs)** containing only the subset of these attributes that are required in order to access a set of data or a Blockchain-related service under the principle of **selective disclosure**. In order to support this and other privacy-achieving properties, we designed an **enhanced variant of the DAA-A crypto protocol** that offers **anonymity**, **unlinkability**, and **unforgeability**, while also being the first of its kind to offer strong guarantees in the integrity of the Wallet when performing attribute attestations.

One of the main contributions of the ASSURED ABE scheme is the provision of the ability for devices to perform **encryption in a decentralized manner**, without requiring a central entity (such as the SCB) to have access to the actual data or the underlying keys. This is a significant contribution compared to similar schemes in the literature, which require a centralized trusted entity to perform key management or to handle the encryption process, since we **eliminate the requirement for placing a large degree of trust in such a centralized entity**. In addition, we have implemented a wide variety of features that enable the ASSURED ABE scheme to fulfill the security and privacy requirements characterizing the industrial domains represented by the envisioned use cases. For example, in cases with strong privacy requirements (such as the *DAEM Smart Cities* use case that involves the handling of sensitive personally identifiable data), the ASSURED ABE scheme enables devices to encrypt a set of data in a manner that does not divulge any information about their identity or their implementation details.

Overall, the TPM-based Wallet and the ABE scheme offered by ASSURED constitute a major step forward in the state-of-the-art regarding secure and privacy-preserving identity management, providing a valuable set of tools that can be used by organizations belonging to a wide

variety of organizations belonging to various industry application domains, characterized by a strict set of security and privacy requirements. In order to verify the fulfillment of such requirements, we have provided rigorous mathematical proofs on both these components, that testify to the correctness of the envisioned properties. In addition, we have provided experimental measurements on the timings of all implemented schemes, in order to demonstrate their efficiency in practical environments.

List of Abbreviations

Abbreviation	Translation
ABAC	Attribute-based Access Control
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
VC	Verifiable Credential
Vf	Virtual Function
VM	Virtual Machine
VP	Verifiable Presentation
Vrf	Verifier
WP	Work Package
ZTP	Zero Touch Provisioning

References

[1] ASSURED Blockchain based Control Services, Sharing Crypto Functions for Decentralized Data Storage, and Access Control Final Version. Assured blockchain architecture. Deliverable D4.4, February 2023.

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

[3] Jan Camenisch, Manu Drijvers, and Anja Lehmann. Anonymous attestation using the strong diffie hellman assumption revisited. In *International Conference on Trust and Trustworthy Computing*, pages 1–20. Springer, 2016.

[4] 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.

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

[6] Liqun Chen, Nada El Kassem, Anja Lehmann, and Vadim Lyubashevsky. A framework for efficient lattice-based daa. In *Proceedings of the 1st ACM Workshop on Workshop on Cyber-Security Arms Race*, pages 23–34, 2019.

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

[8] The ASSURED Consortium. Assured blockchain architecture. Deliverable D1.4, November 2021.

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

[10] The ASSURED Consortium. Assured blockchain-based control services and crypto functions for decentralized data storage, sharing and access control. Deliverable D4.3, February 2022.

[11] The ASSURED Consortium. Assured layered attestation and runtime verification enablers design implementation. Deliverable D3.2, November 2022.

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

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

[14] The ASSURED Consortium. Assured layered attestation and runtime verification enablers design & implementation - version 2. Deliverable 3.3, February 2023.

[15] The ASSURED Consortium. Assured secure and scalable aggregatenetwork attestation - version 2. Deliverable 3.7, February 2023.

[16] The ASSURED Consortium. Updated security context broker specifications and blockchain peer & smart contract. Deliverable D2.2, February 2023.

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

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

[19] Amit Sahai and Brent Waters. Fuzzy identity-based encryption. In Ronald Cramer, editor, *Advances in Cryptology – EUROCRYPT 2005*, pages 457–473, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[20] Syh-Yuan Tan and Wun-She Yap. Cryptanalysis of a cp-abe scheme with policy in normal forms. *Information Processing Letters*, 116(7):492–495, 2016.

[21] Syh-Yuan Tan, Kin-Woon Yeow, and Seong Oun Hwang. Enhancement of a lightweight attribute-based encryption scheme for the internet of things. *IEEE Internet of Things Journal*, 6(4):6384–6395, 2019.

[22] Trusted Computing Group. Trusted Platform Module Library Family "2.0" Specification - Parts 1-4 and Code, Revision 1.59. https://trustedcomputinggroup.org/resource/tpm-library-specification/.

[23] Dan Yamamoto, Yuji Suga, and Kazue Sako. Formalising linked-data based verifiable credentials for selective disclosure. In *2022 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, pages 52–65. IEEE, 2022.

[24] Xuanxia Yao, Zhi Chen, and Ye Tian. A lightweight attribute-based encryption scheme for the internet of things. *Future Generation Computer Systems*, 49:104–112, 2015.

ANNEX

7.1 TPM-based Implementation of Wallet Methods

In Section 4.3, we provided the technical descriptions of the actions and cryptographic operations performed in the context of each of the phases corresponding to methods performed by the TPM-based Wallet. Here, we provide details on the TPM-based implementations of these operations.

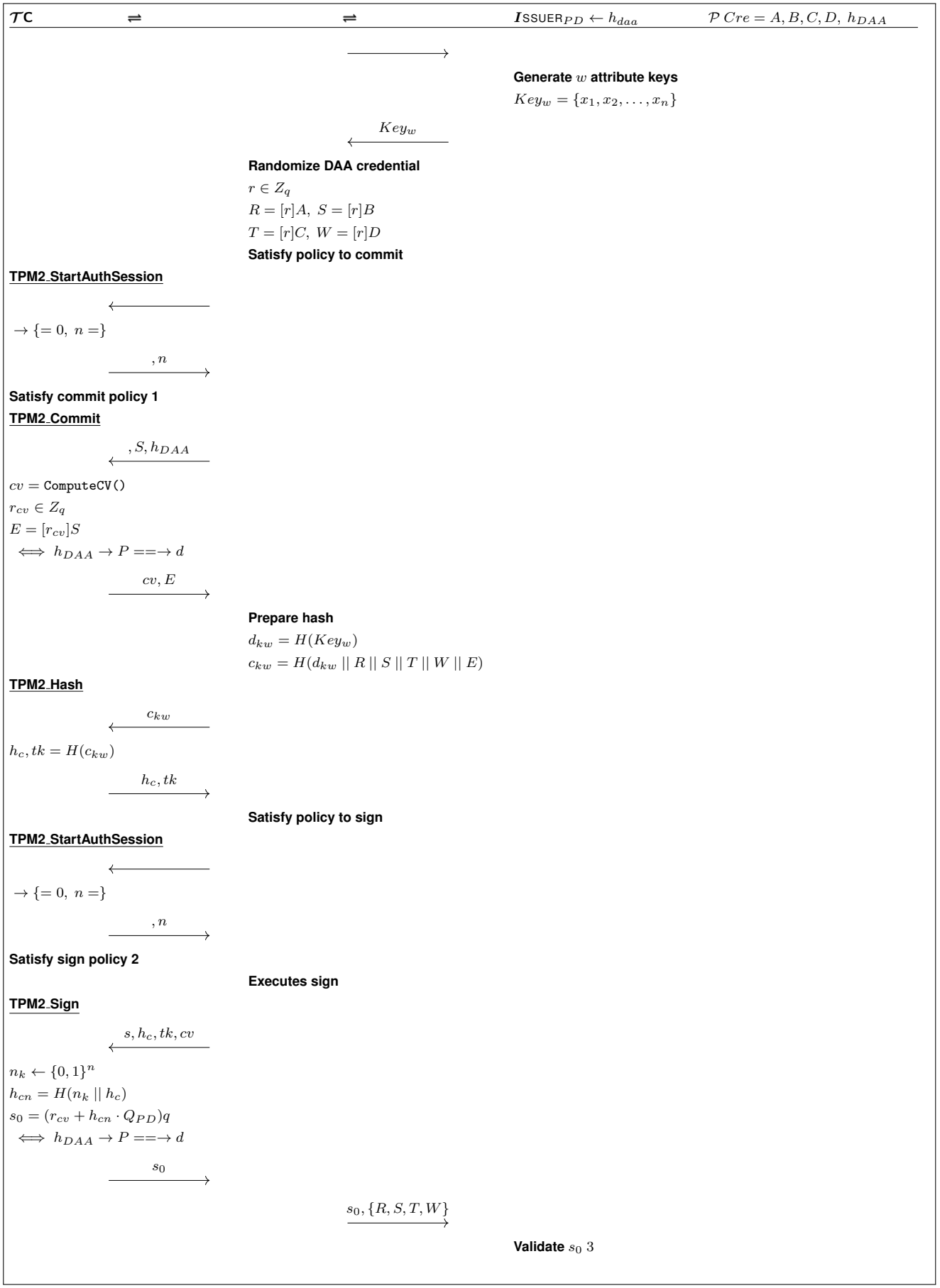

Figure 7.1: Create Verifiable Credential: Verify Wallet Integrity

$\mathcal{TC}$	$\rightleftharpoons$	$\rightleftharpoons$	$\boldsymbol{I}$SSUER$\mathcal{P}$ $Cre = A, B, C, D,\ h_{DAA}$
			Calculates attribute credential
			$\Gamma = + \sum_{k=1}^{k=n} x_k G_k$

Figure 7.2: Create Verifiable Credential: Issue Verifiable Credential

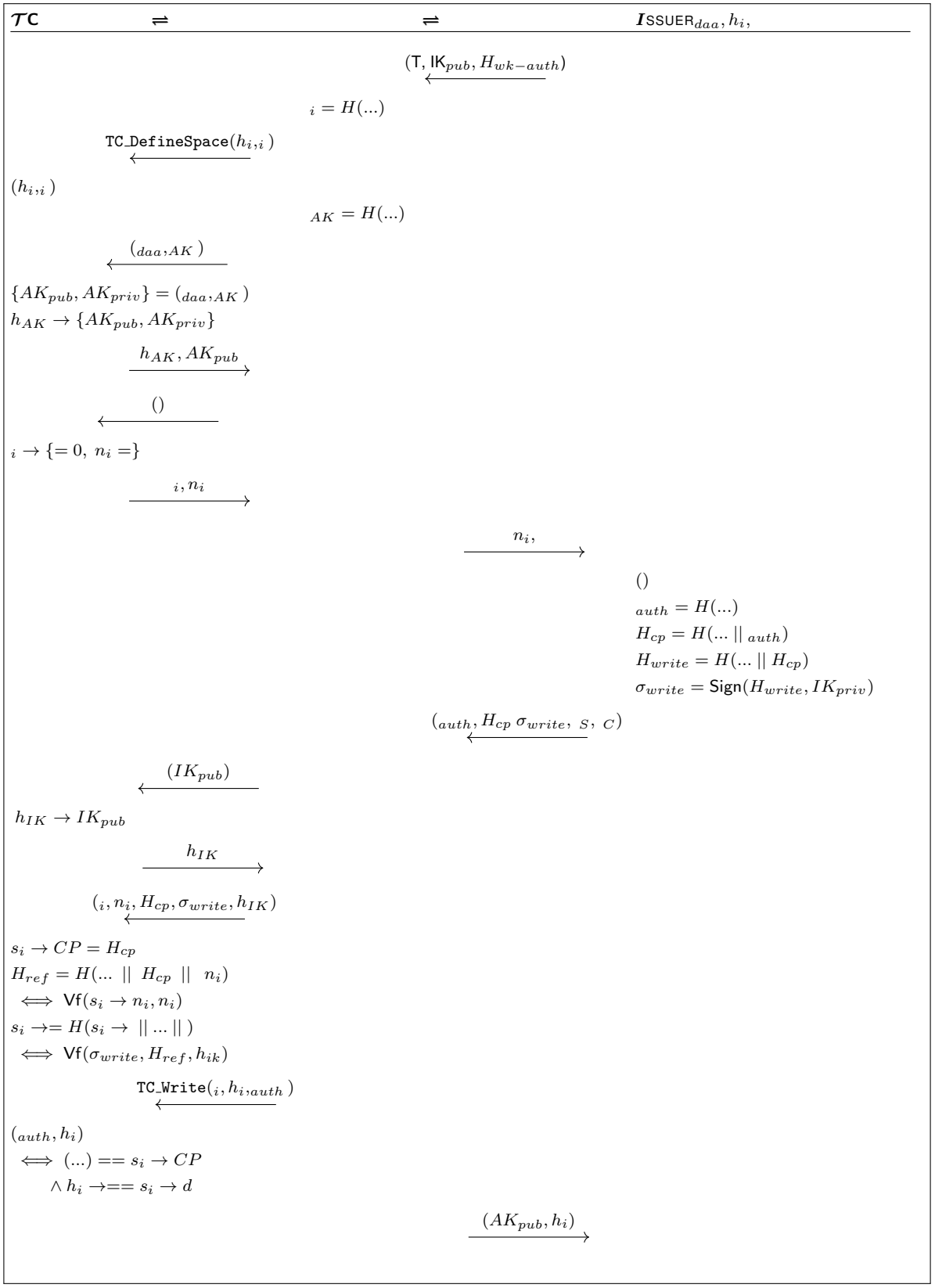

Figure 7.3: Holder Initialization and Registration

$\mathcal{TC}$ $\rightleftharpoons$

$\mathcal{P}\ I_p, s, n$

Satisfies Commit Policy

TPM2_PolicyCommandCode

$$\xleftarrow{\texttt{CC\_Commit}, s}$$

$s \rightarrow d = H(s \rightarrow d \,||\, \texttt{CC\_PolicyCommandCode} \,||\, \texttt{CC\_Commit})$

TPM2_PolicyOR

$$\xleftarrow{\mathcal{P}, s}$$

$s \rightarrow d = H(s \rightarrow d = 0 \,||\, \texttt{CC\_PolicyOR} \,||\, \mathcal{P})$

$\iff s \rightarrow d \in \mathcal{P}$

TPM2_PolicyAuthorizeNV

$$\xleftarrow{I_p, s}$$

$s \rightarrow d = H(s \rightarrow d = 0 \,||\, \texttt{CC\_PolicyCommandCodeNV} \,||\, I_p \rightarrow name)$

$\iff s \rightarrow d \in I_p$

Figure 7.4: Satisfy policy for committing

$\mathcal{TC}$ $\rightleftharpoons$

$\mathcal{P}, \mathcal{T}\ I_p, s, n, WK_{priv}, h_{wk}$

Satisfies Sign Policy

$a = H(n \,||\, 00_{16} \,||\, 00_{16} \,||\, 00_{16})$

$\sigma_a = \texttt{Sign}(a, WK_{priv})$

TPM2_PolicySigned

$$\xleftarrow{\sigma_a, a, s, n}$$

$s \to d = H(s \to d \,||\, \texttt{CC\_PolicySigned} \,||\, h_{wk} \to name \,||\, 00_{16})$

$\iff n \stackrel{?}{=} s \to n \wedge \texttt{Verf}(\sigma_a, a, h_{wk} \to pk)$

TPM2_PolicyPCR

$$\xleftarrow{\mathcal{T}, s}$$

$s \to d = H(s \to d \,||\, \texttt{CC\_PolicyPCR} \,||\, \mathcal{T} \,||\, \texttt{PCR}[t] \to digest\ \forall\ t \in \mathcal{T})$

TPM2_PolicyOR

$$\xleftarrow{\mathcal{P}, s}$$

$s \to d = H(s \to d = 0 \,||\, \texttt{CC\_PolicyOR} \,||\, \mathcal{P})$

$\iff s \to d \in \mathcal{P}$

TPM2_PolicyAuthorizeNV

$$\xleftarrow{I_p, s}$$

$s \to d = H(s \to d = 0 \,||\, \texttt{CC\_PolicyCommandCodeNV} \,||\, I_p \to name)$

$\iff s \to d \in I_p$

Figure 7.5: Satisfy policy for signing