Project Title	AN OPEN, TRUSTED FOG COMPUTING PLATFORM FACILITATING THE DEPLOYMENT, ORCHESTRATION AND MANAGEMENT OF SCALABLE, HETEROGENEOUS AND SECURE IOT SERVICES AND CROSS-CLOUD APPS
Project Acronym	RAINBOW
Grant Agreement No	871403
Instrument	Research and Innovation action
Call / Topic	H2020-ICT-2019-2020 / Cloud Computing
Start Date of Project	01/01/2020
Duration of Project	36 months

D2.2 – RAINBOW Collective Attestation Policy Enablers Design

Work Package	WP2 – Security and Trust for Fog and Cross-Cloud Services
Lead Author (Org)	Thanassis Giannetsos (DTU)
Contributing Author(s) (Org)	Heini Bergsson Debes, Benjamin Larsen (DTU) Panagiotis Gouvas, Konstantinos Theodosiou (UBITECH) Ronald Toegl, Raphael Schermann (IFAT)
Due Date	31/12/2020
Actual Date of Submission	15/02/2021
Version	V1.0

The work described in this document has been conducted within the project RAINBOW. This project has received funding from the European Union's Horizon 2020 (H2020) research and innovation programme under the Grant Agreement no 871403. This document does not represent the opinion of the European Union, and the European Union is not responsible for any use that might be made of such content.

Project No 871403 (RAINBOW)

D2.2 – RAINBOW Collective Attestation Policy Enablers Design

Date: 15/02/2021

Dissemination Level: PU

Dissemination Level

X	PU: Public (*on-line platform)
	PP: Restricted to other programme participants (including the Commission)
	RE: Restricted to a group specified by the consortium (including the Commission)
	CO: Confidential, only for members of the consortium (including the Commission)

Project No 871403 (RAINBOW)

D2.2 – RAINBOW Collective Attestation Policy Enablers Design

Date: 15/02/2021

Dissemination Level: PU

Versioning and contribution history

Version	Date	Author	Notes
0.1	01.09.2020	Thanassis Giannetsos (DTU)	Table of Contents and partner contribution assignment
0.2	30.09.2020	Ronald Toegl (IFAT)	Updated version
0.3	15.01.2021	Ronal Toegl, Raphael Schermann (IFAT)	Content for Chapters 2, 5 and parts of Chapter 7
0.4	20.01.2021	Panagiotis Gouvas, Konstantinos Theodosiou (UBITECH)	Content for Chapter 6
0.5	22.01.2021	Thanassis Giannetsos, Heini Bergsson Debes, Benjamin Larsen (DTU)	Content for Chapters 1, 3, 4 and finalization of Chapter 7
0.6	30.01.2021		First Internal Review
0.8	05.02.2021	Thanassis Giannetsos (DTU)	Addressing Review Comments, added Conclusions
1.0	15.02.2021	Thanassis Giannetsos (DTU)	Finalized Content

Executive Summary

Deliverable D2.2 provides the RAINBOW security, privacy and trust extensions enhanced with **secure remote attestation capabilities for verifying the configurational attestation policies and properties** as well as Direct Anonymous Attestation for privacy-preserving and accountable services. More specifically, it presents a new set of attestation protocols for supporting trust aware service graph chains with verifiable evidence on the integrity assurance and correctness of the comprised fog nodes. It is the first step towards the provision of a **secure overlay mesh network for delivering the high-level functionalities related to secure (edge and mesh) device identification and integrity, data integrity and confidentiality, anonymity and resource integrity** as described in the overall framework reference architecture (see Deliverable D1.2 [21]).

In order to support enhanced **system and network trust assurance**, RAINBOW has defined the security protocols that are necessary for providing a range of **secure attestation services in order to support verifiable evidence on the correct configuration state and/or execution of a remote platform**; ranging from secure boot to run-time integrity referring to the entire lifecycle of the platform. Two enablers of trust are of interest towards protecting and proving the *authenticity* and *integrity* of fog nodes. Whereas integrity provides evidence about correctness, authenticity provides evidence of provenance.

As part of the overall RAINBOW attestation toolkit, the main goal is to allow the creation of **privacy- and trust-aware service graph chains** (managed by the Orchestration Lifecycle Manager and estabslished by the RAINBOW Deployment Manager) through the provision of **zero-touch configuration functionalities**: *fog nodes, wishing to join a fog cluster, adhere to the compiled attestation policies by providing verifiable evidence on their configuration integrity and correctness.* In other words, the framework should provide guarantees that a node will be able to join a network (and participate in the underlying dynamic routing scheme as well as the privacy-preserving key management process) ***if and only if*** it can prove to the Orchestrator ($\mathcal{O}$rc) that it is at a "correct state" - without, however, the $\mathcal{O}$rc needing to know the node's state beforehand. This allows RAINBOW to support the secure enrollment and integration of heterogeneous devices and platform equipped with different computing resources and operating systems.

In terms of design, as will also be described in the following sections, the focus of RAINBOW Enhanced Remote Attestation is on cloud-native component (denote as virtual function, $\mathcal{V}$F) **Integrity Verification** and on **secure enrolment**. Integrity verification is the process by which a fog node (i.e., hosting a $\mathcal{V}$F) can report in a trusted way (at any requested time) the current status of its configuration. It entails the provision of integrity measurements and guarantees during both the **deployment and operation of a** $\mathcal{V}$F; covering the system integrity at the deployment phase by the RAINBOW Orchestrator but also ensuring the integrity of the loaded components during their run-time execution.

Contents

List of Figures

List of Tables

List of Abbreviations

Abbreviation	Translation
AE	Authenticated Encryption
AK	Attestation Key
CA	Certification Authority
CSR	Certificate Signing Request
DAA	Direct Anonymous Attestation
EA	Enhanced Authorization
EK	Endorsement Key
NMS	Network Management System
PCA	Privacy Certification Authority
PCR	Platform Configuration Register
RA	Remote Attestation
S-ZTP	Secure Zero Touch provisioning
TC	Trusted Component
TLS	Transport Layer Security
TPM	Trusted Platform Module
Vf	Virtual Function
VM	Virtual Machine
Vrf	Verifier
WP	Work Package
ZTP	Zero Touch Provisioning

Chapter 1

Introduction

The main goal of this deliverable is to present the first release of the **RAINBOW Collective Attestation enablers** towards fulfilling the main vision of the project for the establishment and management of trust-aware service graph chains. In this context, *the communication over the continuum from edge devices to fog nodes and back end cloud systems must support secure interactions between all participating entities in order to establish **service-specific "fog/edge node communities of trust".*** This is considered as one of the main goals towards **"security and privacy by design"** solutions, including all methods, techniques, and tools that aim at enforcing security and privacy at software and system level from the conception and guaranteeing the validity of these properties.

RAINBOW will achieve high security and privacy guarantees by using a **Trusted Platform Module (TPM)**, enabled through the RAINBOW Sidecar proxy [21], as a central building block for the **trusted exchange of data as well as for secure device management**. TPMs are trust anchors that allow RAINBOW to develop new, highly **efficient attestation techniques** for both device authentication purposes (when joining the IoT deployment) but also continuously attesting edge device integrity and, in doing so, assess the security of involved devices. *The goal is to prove to remote parties that a fog node, its OS and running services are intact and trustworthy.*

To do so RAINBOW is leveraging advanced cryptographic primitives (Direct Anonymous Attestation (Chapter 4) for privacy) and enhanced remote attestation (Chapter 3) for security and operational assurance. At a conceptual level, the goal is to enable fog/edge entities to establish and maintain trust during the entire system life-cycle. *This stems from establishing **roots of trust** in components (by leveraging the attached TPM), and using these roots of trust to establish and maintain trust relationships.*

However, in the road towards the establishment of such trust-aware SGCs, **fog/edge node interaction is a challenge by itself** since the target environment is **dynamic** and **uncontrolled**. More specifically, a RAINBOW deployment consists of nodes that formulate temporal connections. In addition, nodes must route packets to each other without relying on static routing tables and fixed network subnets. On top of that, there are two additional crucial constraints that need to be taken into consideration. The first is the **lack of a network addressing scheme** and the second is the **establishment of trust** among the nodes that participate in the fog deployment towards the creation of "communities of trusted devices" that can enable the secure community communications and can then be used for the trusted deployment of the envisioned services.

Addressing these challenges lies in the heart of RAINBOW towards the provision of a **secure overlay mesh network for delivering the high-level functionalities related to secure (edge and mesh) device identification and integrity, data integrity and confidentiality, anonymity**

and resource integrity as described in the overall framework reference architecture (see Deliverable D1.2 [21]).

*We have to note that in this deliverable the focus is on presenting the **logistics, mode of operation, and workflow of actions** of all the core RAINBOW security, privacy and networking enablers (as standalone components) and not the complete RAINBOW overlay mesh networking stack with all the TPM-based security components integrated*. A description of the conceptual architecture of this overall RAINBOW security solution, along with a detailed documentation of the design choices and its benefits, is presented in Chapter 7; more details on the subsequent modeling and implementation activities will be described in D2.4.

1.1 Scope and Purpose

The main purpose of this deliverable is to **present the first release of the RAINBOW Attestation Toolkit enhanced with integrity verification and DAA trust extensions for attesting configurational properties of a deployed fog node**. More specifically, it documents the complementary functionality of the integrity verification schemes focusing on ensuring not only the trust level of each TPM-equipped platform but also the strong trust relations that must be established among interacting entities. This requires the consideration of different aspects in each case; for instance, trusting a TPM first requires trusting that it operates correctly, and in particular that sequences of TPM commands are executed correctly, while ensuring that the interactions between attested entities is secure is required in ensure to maintain the trust between them.

The goal is to describe the two specific functionalities, Attestation by Proof and Attestation by Quote (see Figure 3.2), for enabling the automatic and secure establishment of trust between deployed devices. The evidence of the integrity state of the configuration properties is authenticated by the attached TPM.

1.2 Relation to other WPs and Deliverables

In what follows, Figure 1.1 depicts the relationships of the deliverable to the other Work Packages (WPs). As aforementioned, the main purpose of this document is to consolidate, formally define and evaluate the RAINBOW secure remote attestation solution and mesh networking stack. The goal is to create a coherent analysis of the three mechanisms produced (Attestation by Proof, Attestation by Quote, and Direct Anonymous Attestation) while also defining how the dynamicity of the solution will function in the context of RAINBOW.

1.3 Deliverable Structure

In the remainder of this deliverable, we provide a detailed documentation of the enhanced RAINBOW attestation enablers that have been designed for supporting platform authentication and integrity verification while enabling the provision of privacy-preserving and accountable services. First, we start by putting forth, in Chapter 2, an overview of the trusted component, namely the Trusted Platform Module (TPM), considered in the context of RAINBOW as the underlying Root-of-Trust coupled with details on the core TPM functionalities considered in our environment towards secure device management and remote attestation. Then, details on the basic building blocks, mode of operation and workflow of actions are provided for the designed RAINBOW trust

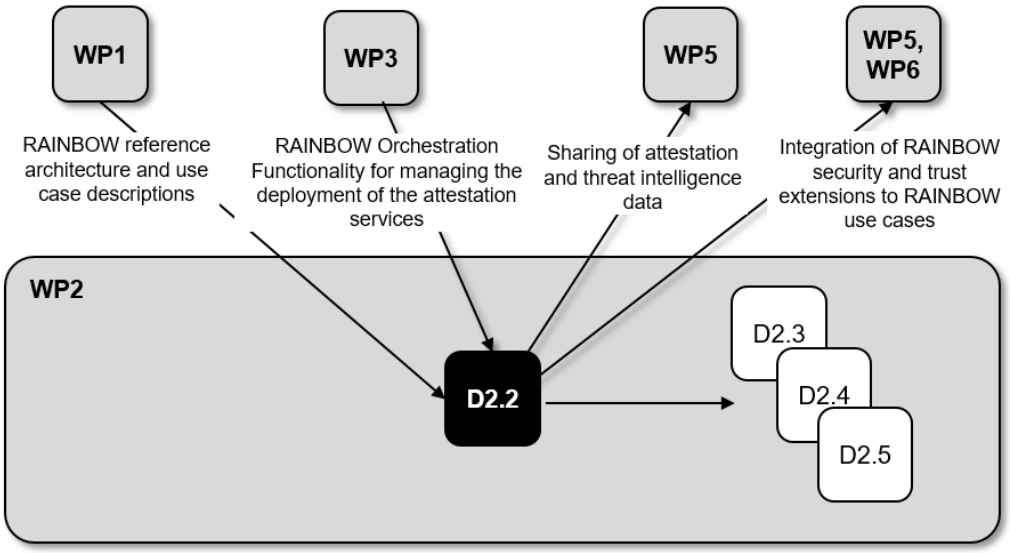

Figure 1.1: Relation to Other Deliverables and Work Packages

extensions; namely Zero-Touch Integrity Verification for verifying the current status of fog node's configuration when trying to (securely) enroll to the fog cluster (Chapter 3), and Direct Anonymous Attestation (DAA) (Chapter 4) for enabling strong privacy guarantees throughout the entire operation of a fog-based ecosystem. These trust extensions are also coupled with a comprehensive mapping of the TPM commands that need to be securely executed by the underlying TPM. Next, we proceed with the description of another RAINBOW security enabler towards the establishment of anonymous secure and authenticated communication channels needed during the node registration and key establishment process (Chapter 5). Finally, we elaborate on the current CJDNS networking mechanism considered in RAINBOW (Chapter 6) and present the vision of a secure overlay mesh network (enhanced CJDNS mesh networking stack) with all the aforementioned trust extensions integrated to be implemented in the upcoming tasks of WP2 (Chapter 7). Chapter 8 concludes the deliverable.

Chapter 2

Hardening the Fog/Edge IoT Stack: Intertrustability of System Composability

2.1 Secure Remote Device Management

In RAINBOW, we are going to implement a **decentralized Root-of-Trust system** with the use of the Trusted Platform Module. Here, the **TPM is the root anchor of the overall system** and described more precisely in Section 2.1.2. The TPM can also be used to **ensure secure boot** which is a crucial requirement for our newly designed trust extensions that attempt to verify the configuration and behavioural correctness of all fog/edge nodes comprising the target fog-based ecosystem. In RAINBOW the **trust mechanism is established on the attestation of PCR quotes in interaction with the TPM**. This inherits the capability for secure boot, i.e., each device that is using this trust mechanism is able to guarantee a secure boot startup. Due the significant implementation overhead of the secure boot startup and variety of supported end devices the implementation of a secure boot mechanism is out of the scope in the RAINBOW project. It also would make no further scientifically contributions regarding the state-of-the-art since there a number of different variants offering such a functionality. This is only interesting in an industrial mass production.

2.1.1 Towards Decentralized Roots-of-Trust

A Root-of-Trust (RoT) is a crucial element in a computer-system and even more in a distributed environment as the ones encounted in RAINBOW. **Roots-of-trust enable other components in a SW system to come to a trust decision for a given situation**. A root-of-trust must be relied on as itself cannot be verified, thus there are high requirements on their correctness and robustness [40, 47]. Nowadays, there exist several concepts to establish a RoT.

One solution is the **Trusted Execution Environment (TEE)**. A TEE is an extra secure environment for execution of sensitive data or safe critical code. This Code ensures a high level of trust. Figuratively speaking its a **secured world with an isolation and access control to the normal world**. The biggest drawback compared to the TPM is that the TEE is not physically isolated from the rest of the processing system, i.e., side-channel attacks (software and physical) [49].

Another solution is the **Device Identifier Composition Engine (DICE)** architecture [44] specified by the TCG. It was designed for resource constrained devices, i.e., that are not capable of including a TPM. However, according to TCG DICE does not imply any reduced implementation benefits for complex systems. Compared to the TPM here DICE does not support the complex DAA Protocol

used for privacy-preserving node communication (Chapter 4). Also side-channel attacks can occur and DICE has not the same certified security level as the TPM. A secure, distributed system such as the RAINBOW platform needs a trust anchor that verifies and ensures the correctness of the overall system and provide an efficient mechanism to prevent side-channel attacks. **For RAINBOW a decentralized trust anchor has been chosen, in the form of Trusted Platform Modules on the system's end points.** This anchor or main building block is a Trusted Platform Module, short TPM. A TPM is a cryptographic co-processor on an computer system [8] and is specified by the Trusted Computing Group [5]. There are several possibilities to use a TPM. According to the TCG [2] basically four types of TPM can be used (lowest number is the highest security level).

1. **Discrete TPM**. The TPM provides the highest level of security and uses a discrete chip that provides only cryptographical functionality. At this level the chip is designed to be tamper-resistant.
2. **Integrated TPM**. Also the teram hardware TPM is used. However, beside cryptographical functionality the chips also executes non-security related software.
3. **Firmware TPM**. The TPM is implemented in protected software, e.g., Trusted Executed Environment (TEE), and runs on the core CPU.
4. **Software TPM**. The TPM is implemented as a emulator. Furthermore, it does not ave any key applications and therefore no secure storage for the keys. This purpose is optimal to build a prototype.

There also exists a virtual approach called virtual TPM. In this case, the TPM provides for each virtual machine a software TPM which offers the same command set as a physical TPM. *How virtual TPMs protect keys and data is implementation specific and depends on the operator's policies.*

Due the high risk of side-channel attacks, at least a hardware TPM shall be used when the computer device is physically accessible, otherwise a firmware or even a software TPM may be used. Subsection 2.1.3 covers the core TPM functionalities of interest (the ones based upon the RAINBOW trust extensions have been designed) in more detail.

The distributed use of TPMs is a strong enabler for secured decentralized systems. In this context, decentralized deployment of roots-of-trust offers several benefits in contrast to single centralized ones. First, the system is more tolerant against faults and no single point of failure is introduced. As an example: In a mesh network one node fails, the network algorithm re-routes the other participants to ensure the availability of the other nodes. Second, such a system provides more resistance against attacks, i.e., no single attack point that demilitarize the entire system. Third, it increases the collusion resistance in the way that participants are not able to collude for own prioritized targets.

Figure 2.1 shows an attestation example. The attestation is used to confirm the identity of the platform and prove to a remote party that the Operating System (OS) is in an valid state. As it will be described in Chapter 3, this functionality is the base point used in our newly designed **RAINBOW Configuration Integrity Verification trust extensions towards the establishment of trust-aware service graph chains (SGCs)**. Here, the root-of-trust is the OPTIGA™ TPM chip from Infineon. The Application within the platform generated a public and private key-pair

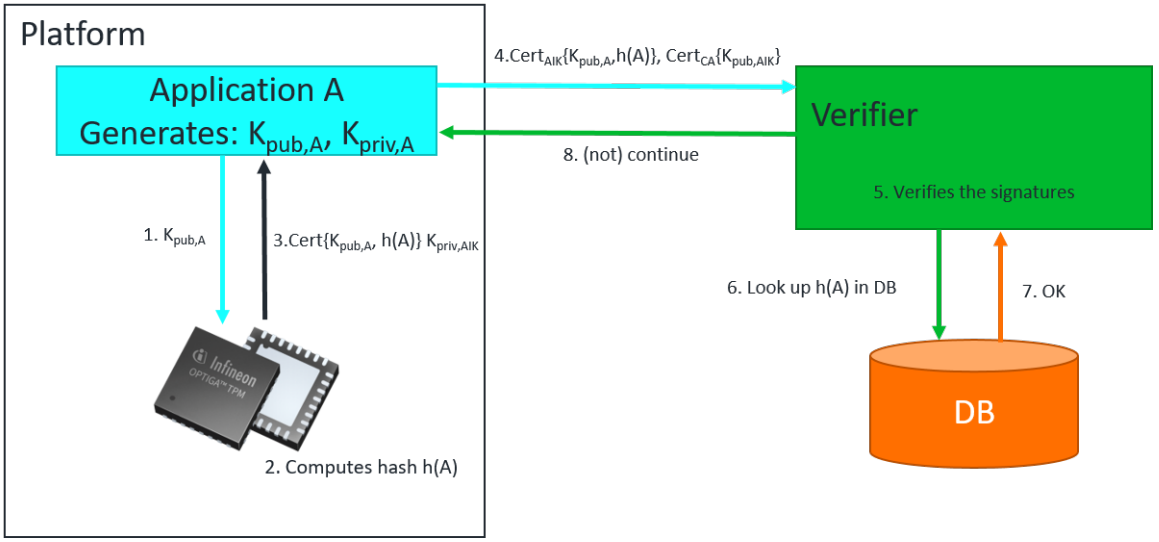

Figure 2.1: Remote Attestation Example (based on: [11],p.3)

and sends the public portion to the TPM (1). Then the TPM computes a hash out of the message and creates a certification including the public key from A and the generated hash with the private Attestation Integrity Key (AIK) from the TPM (3). To authenticate the platform to a remote party the application send this cert package together with the certificate issued to the TPM by a CA (Certificate Authority) (4). Now, the remote party is able to verify the certificate chain (5) and looks up the hash in a database (6). If the certificate chain is trustworthy the communication continues, e.g., establish a session key, otherwise the platform will be informed that he is not able to continue further (8). This, or a similar attestation protocol can be executed between each node of the RAINBOW network, potentially without communicating with a central instance.

2.1.2 Rainbow Hardware Security Anchor: Trusted Platform Module

As aforementioned, in RAINBOW we are going to use a Trusted Platform Module (TPM). More specifically, we are using the OPTIGA™ TPM SLx 9670 TPM2.0 products. These are fully TCG compliant TPM products with CC (EAL4+) and FIPS certification. According to [34] the in Rainbow used SLB 9760 includes following noteworthy features:

- SPI (Serial Peripheral Interface) with a transfer-rate $\leq$ 43 MHz
- RNG according to [12]
- EK and EK certiciate are fully personalized
- 24 PCRs. Here, SHA-1 or SHA-256 is supported.
- $>$ 696 bytes free NV memory
- $\leq$ 3 loaded sessions (TPM_PT_HR_LOADED_MIN)
- $\leq$ 64 active sessions (TPM_PT_ACTIVE_SESSIONS_MAX)
- $\leq$ 3 loaded transient Objects (TPM_PT_HR_TRANSIENT_MIN)
- $\leq$ 7 loaded persistent Objects (TPM_PT_HR_PERSISTENT_MIN)

- $\leq$ 8 NV counters
- $\leq$ 1 kByte for command parameters and response parameters
- $\leq$ 768 Byte for NV read or NV write
- 1420 Byte I/O buffer

The power-management is handled by the TPM internally, i.e., no specific power-down or standby mode. Here, the TPM invokes a low-power state after a command/response transaction. The SPI interface is configures with wake-up, i.e., TPM wakes up when a transaction is started on the bus.

2.1.3 Rainbow Software: Trusted Platform Module as a Building Block

As mentioned before, a Trusted Platform Module (TPM) is a security controller for cryptographic devices and is physically separated from the main processor, i.e., cryptographic co-processor. It is designed to protect security critical data, e.g., cryptographic keys, passwords, and to resist logical and physical attacks. The TPM includes the following core concepts which are important in context to RAINBOW [8, 40]:

- **TPM Binding.** Here, the TPM encrypts data with a unique RSA key. This key is derived from a storage key.
- **Platform Configuration Registers (PCRs).** These registers are used to cryptography measure the software state or the software configuration. As an example a boot process: firmware $\rightarrow$ firmware $\rightarrow$ peripheral firmware $\rightarrow$ boot loader $\rightarrow$ kernel $\rightarrow$ drivers $\rightarrow$ libraries $\rightarrow$ applications $\rightarrow$ security settings. If one of a PCR value does not math a certain behavior the device will not boot.
- **TPM Sealing.** Here, the TPM encrypts data under a non-mitigable key and to a set of PCR values.
- **Attestation.** Attestation is one of the crucial services of a TPM. It is the process by which a platform reports in a trusted way the current status of its configuration. The report can include as much information as required. The basis of the attestation are the measurements recorded in PCRs. They can then be read to know the current status of platform and be also signed to provide a secure report. The signed message can then be sent to the client. It is worth noticing that the TPM does not check the measurements, that is, it does not know whether a measurement is trustworthy or not. The trustworthiness of the measured value comes when an application uses some PCR value in an authorization policy, or remote clients ask for an attestation of some value, and later they evaluate its trustworthiness. Attestation enables such clients to confirm whether the platform has been compromised. Additionally, the TPM offers means of certifying and auditing the properties of keys and data that cross the TPM boundary. Furthermore, it is used to support pseudonyms in relation to DAA.
- **TPM Signing.** Here, enables to sign a signature with a key that is protected within the TPM.

The TPM provides **three persistent key-hierarchies**, namely **platform-, endorsement-, and storage-hierarchy and one volatile hierarchy** called NULL hierarchy. Figure 2.2 shows the hierarchies that will be leveraged within RAINBOW for storing the needed keys.

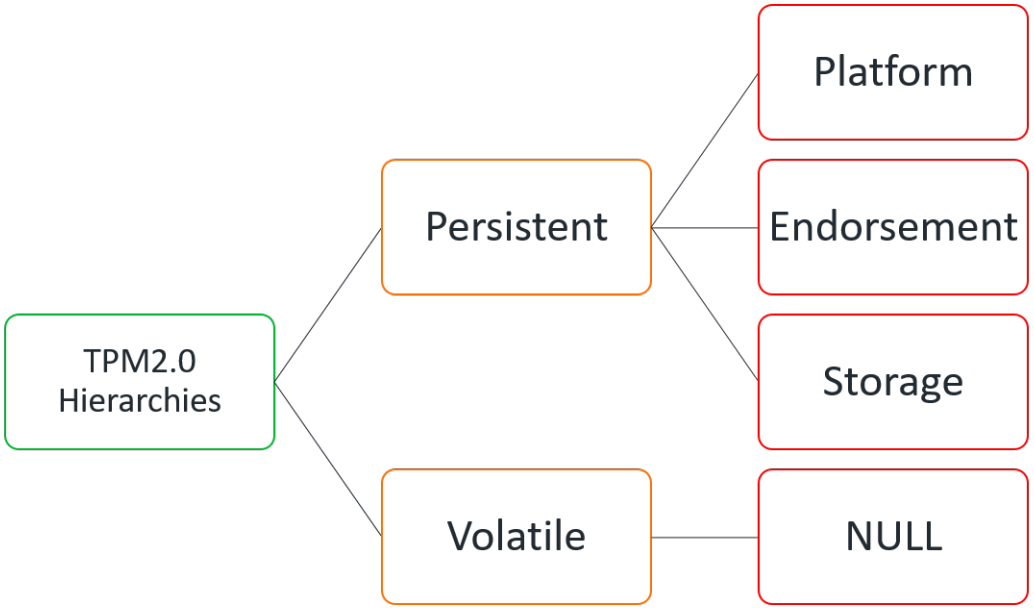

Figure 2.2: TPM2.0 Hierarchies

The Platform hierarchy is under th control of the manufacturer while the Storage hierarchy is used by the platform owner and it can be disabled by the platform owner. Here, the storage hierarchy is designed for non-privacy-sensitive operations. Last, the endorsement hierarchy that is arranged for privacy-sensitive operations. Furthermore, the TPM offers one non-persistent hierarchy called NULL hierarchy. At this hierarchy [8].

To use the current TPM2.0, there exists two different implementations. The first one is the tpm2-software implementation from the Frauenhofer SIT [4] and is based on the TCG specification [5] while the second one called IBM's TPM 2.0 TSS from Ken Goldman [3]. Here, the IBM's TPM 2.0 TSS is not API compatible with the TCG specification but the functionality is equivalent.

Figure 2.3 shows a sequence diagram of how a primary and a child key are generated and saved to the NV-memory inside the TPM2.0. The example is based on the Frauenhofer TSS implementation and its Enhanced System API (ESAPI) layer according to the TCG specification [45]. Beside this TPM functionality, the diagram also includes disk storage mechanism, i.e., keys do not need to be generated again. This general mechanism can be used for all kinds of cryptorpahic keys in the RAINBOW platform; **for each key, specific policies can be added to ensure their correct and consistent usage**.

In step one, the platform needs to initialize the TPM Command Transmission Interface (TCTI) [46] and the TPM startup in the setup-phase. Here, the TCTI interface is needed to transmit and receive TPM commands (Tss2_TctiLdr_Initialize) and the two following Esys commands are used to initialize and startup the Esys context (Esys_Initialize) and to startup the TPM (Esys_Startup). After the setup phase the platform is ready to communicate with the TPM. The usecase in this figure is specific to a key-creation process with persistent key storage. In the Create Primary and store it into the Non-Volatile (NV) memory section. The platform defines some authentication block and sets the authorization value for a resource (here for the primary key) with the function Esys_TR_SetAuth,i.e., access control. Then the platform triggers the Esys_createPrimary function with a predefined key-scheme to create a primary key. This key can later used (create child key section) to derive new child keys under this primary key. With Esys_EvictControl the platform flushes the previous created primary key into the NV memory section, i.e., the key is available after a reboot/shutdown. The next section (load the NV key) serializes the metadata

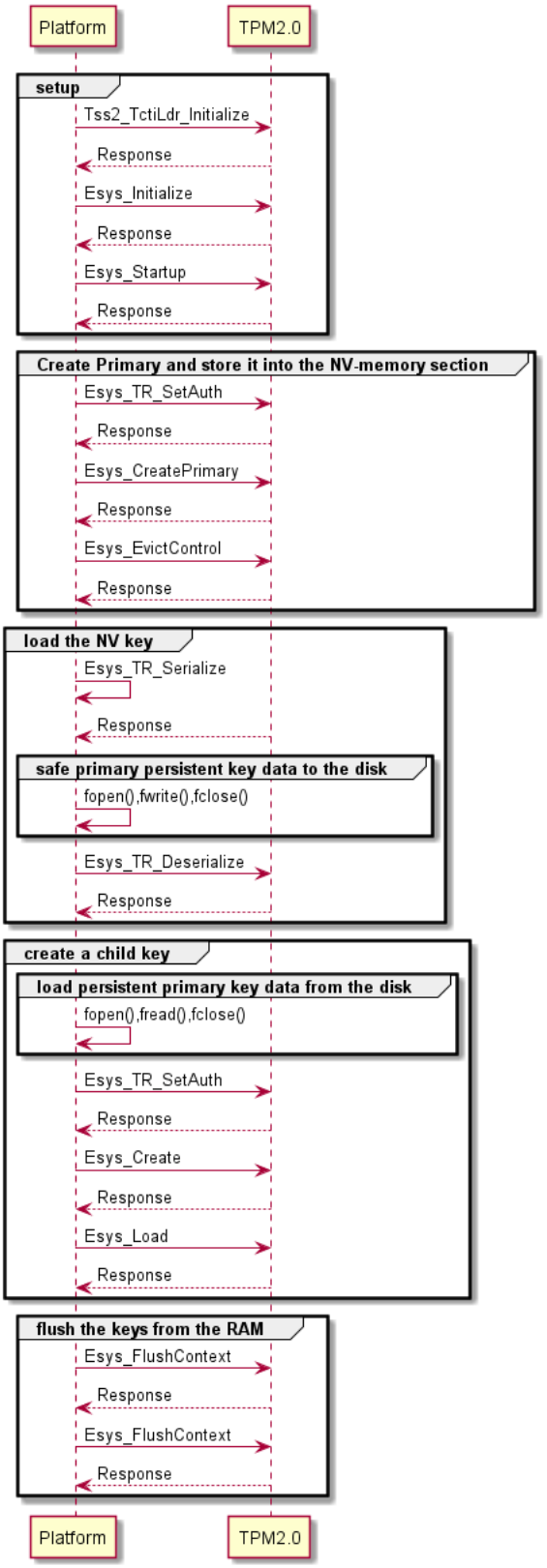

Figure 2.3: Evict Control with disk storage usage

of a TR-Object into a byte buffer for writing it into a file (Esys_TR_Serialize) and stores it onto the hard disk. Note: the deserialize step is additionally. The fourth block is the creation of a child key with the previously stored key data from the hard disk. Here, the platform reads the

key-file form the disk storage, sets as in the create primary key an own authentication value, creates a new cryptographic key (Esys_Create) and loads the key into the RAM (Esys_Load). Note: Esys_CreatePrimary int contrast to Esys_Create implicitly include the load mechanism. Additionally, the fresh generated child key can be also stored in the NV memory section with the Esys_EvictControl command. This depends on the usecase. The last block flushes the key from the RAM to ensure that new keys can be generated. Note: Three loaded keys are storeable in the RAM (tpm2-tool command *tpm2_getcap properties-variable* [4]) For the IBM's TPM 2.0 TSS stack the concepts is the same but the function calls vary. Here, the TSS_Execute function and a specific TPM_CC command-code is called to communicate with the TPM after a TSS_Create function that creates a TSS context. If this context needs to be changed during runtime, this can be done with the TSS_SetProperty function.

In the context of RAINBOW, we are planning to use the IBM's TPM 2.0 TSS stack. For the DAA functionality we are going to use and extend the implementation from Wesemeyer et al. [50]. This implementation is based on the previously mentioned software IBM's TPM 2.0 TSS stack.

2.2 RAINBOW Security Asset Management Services

In this section we want to clarify the need for **Security Asset Management** in the Rainbow Platform. **The assets that we need to support through such a functionality are keys, passwords, user-data, and routing table information for the underlying mesh networking mechanism (i.e.,CJDNS)**.

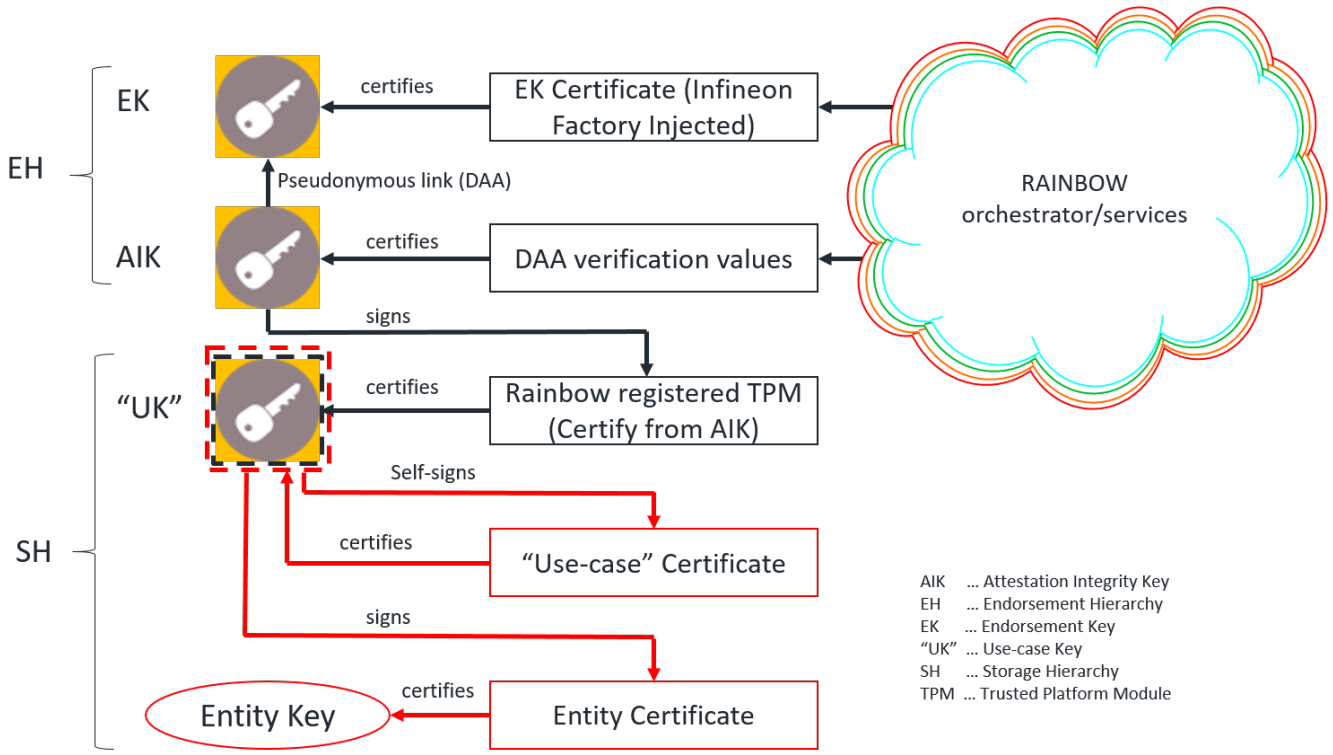

Figure 2.4: Interactive Rainbow Certification Chain

Throughout the Cloud and Fog Computing components of the RAINBOW architecture, **the security assets can be protected by adherence to best practises in data management and software security**. However, on the edge devices, or the IoT devices, **hardware security will be**

employed to a) **ensure interactive, online authentication of the devices and their communications**, and b) **protect keys and data at rest**, i.e. in the time-frame the application software cannot protect it. All this is to be achieved with full privacy protection.

A number of devices, hardware features and software services need to interact to achieve authentication and data protection. We now outline how this can be achieved in the RAINBOW architecture by discussing an example; the interactive, privacy-preserving authentication of the services offered by an abstract "Entity" (e.g. IoT device, edge node,) service.

Figure 2.4 shows the Interactive Rainbow Certification Chain with the key handling. Here, the black lines are in direct interaction with the TPM module and the red ones are managed by the orchestrator with the help of the TPM. The diagram includes two key-storage hierarchies from the TPM, namely the Endorsement-Hierarchy (EH) and the Storage-Hierarchy(SH) described in 2.1.3.

The Endorsement Key is used in the RAINBOW enrollment phase to identify the hardware as member of the RAINBOW ecosystem. The public part of the EK is registered in a one-time enrolment action. The TPM manufacturer vouches for the authenticity and uniqueness of each EK and that is well-protected by a certified TPM. Roge TPMs (for instance if a host device has been stolen) may be revoked with appropriate revocation services (Section 4.3.4).

Once that TPM is registered, an arbitrary number of pseudonymous Attestation Identity Keys can be created by each TPM (Section 4.3.2). **The keys that are needed for the anonymous attestation are located in the EH.** Here, the AIK key is a pseudonymous link to the endorsement key. The AIK is established in the DAA Join Phase and certified by the DAA verification values (Chapter 4). For each AIK, the fact that it is hosted and protected on a certified and registered TPM can be proven with the DAA protocol; the resulting verification values certify that the AIK originates from a RAINBOW TPM, but it gives no indication from which specific device it is from. The lifecycle of the AIK is not restricted; depending on each user services' requirements, it may be created once, in the beginning of the device/software lifecycle, or even at every single execution of a given protocol.

Any User Key can now be either created within the TPM or in a user application. The AIK can be used to sign a User Key. This provides a pseudonymous prove, that UK is part of the Rainbow ecosystem, without unveiling on which platform. In our example, the user Key has two sign functionalities. First self-sign the use-case certificate that is provided by the Orchestrator ($\mathcal{O}$rc) and the second one is to sign the entity certificate for certifying the Entity Key. The entity certificate can now be a conventional cryptographic certificate with an arbitrary lifetime, depending on the service offered. The achieved security level depends on the service configuration. In the example, UK is TPM-protected, but Entity Key is not.

To sum up, **TPMs give full control on which devices join the RAINBOW platform** (Chapter 3), and at the same time allow the services full control whether to make the link to a device visible or not. Services can enjoy TPM protection of their assets, but are not restricted to TPM functionalities.

2.3 Solidifying a System's Integrity: Inter-Trustability of Internal Configuration Properties

A combination of the aforementioned concepts is of great interest to the secure composability of fog-based service graph chains (SGCs), encompassing a broad array of mixed-criticality ser-

vices and applications. In particular, RAINBOW strives to enable orchestration of heterogeneous platforms containing mutable configurations by leveraging the profoundness of existing attestation techniques. In what follows, we elaborate on the inherent functionalities of a TPM that are leveraged by the new set of remote attestation algorithms presented in Chapter 3.

Monotonic Counters for Trusted Measurements. Internally, each TPM has several PCRs that can be used for recording irreversible measurements through accumulation, e.g., extending PCR slot i with measurement m, the TPM accumulates: $PCR_i = hash(PCR_i||m)$. This is an indispensable property towards the creation of strong and transitive roots of trust. For instance, to enforce and regulate trustworthiness of the system boot sequence we can require that all components measure and verify their successors by the following recurrence construct [42]: $I_0 = true; I_{i+1} = I_i \wedge V_i(L_{i+1})$, where $i \leq n \wedge n \in \mathbb{N}^*$, I_i denotes the integrity of layer i and V_i is the corresponding verification function which compares the hash of its successor with a trusted reference value. For example, as in [25], let us assume that we require the boot sequence: $seq\langle sinit, BL(m), OS(m), VS(m), VM(vf), APP(vf)\rangle$, where $sinit$ is the value that the PCR is reset to. If we know that the sequence will yield PCR extensions with the values $v_1, \ldots, v_n$, and all components extend PCR j, then we will trust the chain *if and only if (iff)* PCR$_j = hash(\ldots(hash(sinit||hash(v_1))||hash(v_2))\ldots||hash(v_n))$.

Attestation & Policy-Based Sealing/Binding. Attestation can be either *local* or *remote*. Local attestation is based on Attestation Keys (AKs), which are asymmetric key pairs $AK = \{AK_{pub}, AK_{priv}\}$. To perform local attestation, we enforce usage restrictions (authorization policies) onto AK_{priv}, such as requiring that PCRs must be in a certain state to permit signing operations, e.g., PCR$_j$ (from the example above) actually reflects the accumulation of $v_1, \ldots, v_n$. Thus, using AK_{priv} to sign a nonce chosen by $\mathcal{V}$rf provides indisputable evidence that the machine state is correct. Remote attestation is delegating the verification of PCR$_j$ to $\mathcal{V}$rf, through TPM quotes comprising a signed data structure of the nonce and the contents of a specified choice of PCRs, which $\mathcal{V}$rf verifies against trusted reference values. Regardless of the attestation method, $\mathcal{P}$rv must also prove authenticity to $\mathcal{V}$rf. The TPM contains several key hierarchies, but authenticity is founded specifically in the *endorsement hierarchy*. The root endorsement seed, from which Endorsement Keys (EKs) are generated, passes irrefutable evidence to the EK in a transitive manner. The credibility of the seed, and hence loaded EKs, is usually based on the trustworthiness of the Original Equipment Provider (OEP), which during manufacturing signs, loads, and later vouches that the seed corresponds to a valid TPM [51].

Chapter 3

RAINBOW Zero Touch Configuration: Integrating Trust Extensions into Fog/Edge Secure Enrollment

Leveraging cryptographic techniques and Trusted Components (TPMs) towards protecting and proving the authenticity and integrity of fog nodes is one of the core objectives of RAINBOW. As has been described in previous deliverables [22], *this serves as the foundation on which cloud-based services can start building a well-rounded cybersecurity strategy*.

In order to support enhanced **system and network trust assurance**, RAINBOW has defined the security protocols that are necessary for providing a range of **secure attestation services in order to support verifiable evidence on the correct configuration state and/or execution of a remote platform**; ranging from secure boot to run-time integrity referring to the entire life-cycle of the platform. Two enablers of trust are of interest towards protecting and proving the *authenticity* and *integrity* of fog nodes. Whereas integrity provides evidence about correctness, authenticity provides evidence of provenance.

As part of the overall RAINBOW attestation toolkit [21], the main goal is to allow the creation of **privacy- and trust-aware service graph chains** (managed by the Orchestration Lifecycle Manager and estabslished by the RAINBOW Deployment Manager) through the provision of **zero-touch configuration functionalities**: *fog nodes, wishing to join a fog cluster, adhere to the compiled attestation policies by providing verifiable evidence on their configuration integrity and correctness.* In other words, the framework should provide guarantees that a node will be able to join a network (and participate in the underlying dynamic routing scheme - Chapter 6 - as well as the privacy-preserving key management process - Chapter 4) ***if and only if*** it can prove to the Orchestrator ($\mathcal{O}$rc) that it is at a "correct state" - without, however, the $\mathcal{O}$rc needing to know the node's state beforehand. This allows RAINBOW to support the secure enrollment and integration of heterogeneous devices and platform equipped with different computing resources and operating systems.

The failure of such an attestation process may be the indication of a zero-day vulnerability (or another detected exploit) and/or malfunction, thus, resulting in prohibiting the on-boarding of the target node in the fog cluster and already deployed service graph chain. If any of the enforced security policies fail, from malicious intent or faulty behaviour, the next logical step is to identify what was the cause of this event. This will enable **better situation awareness adaptation** for re-calculating the overall risks and threats of the entire ecosystem (considering the newly identifed vulnerability) allowing **policy adjustments and the compilation of updated mitigation strategies and attestation policies**.

The goal is to observe, model and monitor not only the trust level of each TPM-equipped fog node but also the strong trust relations that must be established among interacting entities. This requires the consideration of different aspects in each case; for instance, trusting a TPM first requires trusting that it operates correctly, and in particular that sequences of TPM commands are executed correctly, while ensuring that the interactions between attested entities is secure is required in ensure to maintain the trust between them. Thus, the best approach - as has been adopted by RAINBOW - is to use a combination of remote attestation protocols towards achieving both **load- and run-time integrity of a device's execution: The assurance that a device works correctly after loading is known as load-time integrity, while run-time integrity refers to the whole process lifecycle.**

Notably, as aforementioned in Chapter 2 secure remote attestation is one of the most popular services provided by the TPM towards the creation of chains-of-trust based on verifiable evidence about the integrity and execution correctness of a platform. **As part of the RAINBOW Attestation Toolkit, we will provide new instances of Platform Integrity Verification and Direct Anonymous Attestation (DAA) (Chapter 4) as platform authentication mechanisms that enable the provision of privacy-preserving and accountable services.**

In terms of design, as will also be described in the following sections, the focus of RAINBOW Enhanced Remote Attestation is on cloud-native component (denote as virtual function, $\mathcal{V}$F) **Integrity Verification** and on **secure enrolment**. Integrity verification is the process by which a fog node (i.e., hosting a $\mathcal{V}$F) can report in a trusted way (at any requested time) the current status of its configuration. It entails the provision of integrity measurements and guarantees during both the **deployment and operation of a** $\mathcal{V}$F; covering the system integrity at the deployment phase by the RAINBOW Orchestrator but also ensuring the integrity of the loaded components during their run-time execution.

3.1 System Model

The considered system (Figure 3.1) is composed of a virtualized network infrastructure in which the Orchestrator ($\mathcal{O}$rc) spawns and governs a set of heterogeneous $\mathcal{V}$F instances, as part of dedicated service graph chains. Each graph is composed of the ordered set of $\mathcal{V}$Fs that the service runs to manage better the correct execution of the onboarded (safety-critical) application workloads and guarantee its offered attributes (e.g., reliability, availability, performance). State-of-the-art software engineering trends are based on the $\mathcal{V}$F microservice concept for achieving high scalability and adequate agility levels [35]. In RAINBOW, we assume the integration of lightweight virtualization techniques, namely *containerization* [10], where applications are decomposed into a mesh of cloud-native containerized $\mathcal{V}$Fs, each one with specific and "small-scope"-stateless processing objectives, packaged on independent virtual execution environments equipped with highly secure anchors (i.e., TPMs) that serve as our RoTs. Each deployed $\mathcal{V}$F contains workload configurations, such as its software image, platform configuration information, and other binaries (see Definition 1), which are measured and securely accumulated into the PCRs of the TPM.

Definition 1 (Configurations)**.** *The configuration set of a* $\mathcal{V}$F *encompasses all objects (blobs of binary data) accessible through unique file identifiers.*

More formally, the $\mathcal{O}$rc maintains a Service Forwarding Graph ($\mathcal{SG}$), of function chains, defined as $\mathcal{SG} = \{s_1, s_2, \ldots, s_n\}$, where $n \in \mathbb{N}^*$. Each service chain comprises a set of deployed $\mathcal{V}$Fs, $s_i = \{vf_1, vf_2, \ldots, vf_m\}$, where $m \in \mathbb{N}^*$ and $s_i \in \mathcal{SG}$, deployed over the substrate virtualized network.

Table 3.1: Notation used

Symbol	Description
$\mathcal{V}$F	A Virtual Function $\mathcal{V}$F
$\mathcal{A}$dv	An adversary resident in a $\mathcal{V}$F
$\mathcal{T}$C	Trusted Component
EK	Endorsement Key containing a public and private part (EK_{pub} and EK_{priv}) and a protected symmetric key used for encrypting child keys (EK^{sk}_{priv})
AK	Attestation Key
σ	Cryptographic signature
KH	Key handle to a loaded key in the $\mathcal{T}$C
$\mathcal{I}$	Selection of PCR identifiers
n	Randomly generated nonce
S	Internal session digest in $\mathcal{T}$C
$A_{\texttt{tmp}}$	Key template
$h_{\texttt{Conf}}{}^{\dagger}$	Expected configuration (PCR hash)
$h_{\texttt{Pol}}$	Policy digest based on $h_{\texttt{Conf}}$
$h_{\texttt{Create}}$	Key creation hash, w. $\mathcal{T}$C state and parent key
$T^{\dagger}$	Creation ticket proving origin of creation hash
A_{cert}	Key creation certificate
$Q_{\texttt{Cert}}$	Quote Certificate
$h_{\beta}{}^{\dagger}$	Hash of a binary

† We further use a prime to denote a reference, e.g., $h'_{\texttt{Conf}}$ is a calculated reference to the actual hash of the PCR contents.

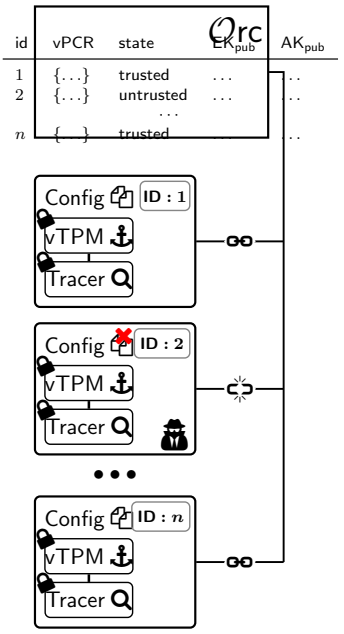

Figure 3.1: Orchestration of Segregated $\mathcal{V}$Fs

The ownership of the physical resources, over which the secure deployment and placement of these SGs take place, is not of interest. Each $vf_i \in \left\{\bigcup_{j=1}^{n} \mathcal{SG}_j\right\}$ is defined as a tuple of the initial form: $vf_i = (id, vPCR, state, EK_{pub}, AK_{pub})$, where id is the unique $\mathcal{V}$F identifier, $vPCR$ refers to an artificial set of PCRs that reflect the obligatory policy (measurements) that must be enforced in the actual PCRs of the target $\mathcal{V}$F, $state$ denotes whether the $\mathcal{V}$F is considered trusted or not (policy-conformant), EK_{pub} and AK_{pub} are the public parts of the EK and AK of the vTPM that is uniquely associated with vf_i.

In addition, each $\mathcal{V}$F is equipped with a Runtime Tracer ($\mathcal{T}$rce) for recording the current state of the loaded software binary data (during both boot-up time and system execution) to be then securely accumulated into the PCRs of the hosted TPM. Tracing techniques are used to collect statistical information, performance analysis, dynamic kernel or application debug information, and general system audits. In dynamic tracing, this can take place without the need for recompilation or reboot. In the context of RAINBOW, a detailed dynamic tracing of the kernel shared libraries, low-level code, etc., and an in-depth investigation of the $\mathcal{V}$F's configuration is performed to detect any cheating attempts or integrity violations. Such a $\mathcal{T}$rce can be realized either as: (*i*) a static binary analyzer for extracting hashed binary data measurements (i.e., digests) [6], or (*ii*) a general, lightweight tracer with kernel-based code monitoring capabilities.

This process builds on top of the IMA feature [43] and records measurements of the $\mathcal{V}$F's software binary images of interest (as specified in the deployed security/attestation policy) that reflect its state/integrity: these can span from *hardware-related properties* related to the BIOS/UEFI and kernel information, to *dynamic properties* such as executable code, structured data and temporary application data (e.g., configuration files, file accesses, kernel module loading). When a measurement is extracted (Section 3.3), a register of the TPM accumulates the digest of the captured event data to protect the integrity and constitutes the basis of the subsequent verification of a $\mathcal{V}$F's trusted state: The trust state is the result of the remote attestation functionalities

of RAINBOW, in which the measurements of the software loaded on a $\mathcal{V}$F is verified either locally (Attestation by Proof) or by the $\mathcal{O}$rc (Attestation by Quote) against reference values that characterize known (and, thus, trusted) software configurations.

Definition 2 (Tracer, $\mathcal{T}$rce)**.** *Given an object identifier (see Definition 1), the* $\mathcal{T}$rce *utility returns (in a secure way) the corresponding object's binary data.*

3.2 High-Level Overview

The solution can be either applied separately to each deployed $\mathcal{V}$F, equipped with a "Root-of-Trust" security anchor (TPM), or the entire Service Graph Chain (SGC). The overall goal is to design secure remote attestation services capable of achieving the security properties of $\mathcal{V}$F **Configuration Correctness, SGC Trustworthiness, Attestation key Protection, Immutability, and Liveness & Controlled Invocation**, as decscribed in Deliverable D2.1 [23].

A key challenge, in this context, is to establish and manage trust between entities, starting from bi-lateral interactions between two single system components and continuing as such systems get connected to ever larger entities. *But how can we make sound statements on the security and privacy properties of single systems and transfer this to statements on the security properties of hierarchical service graph chains?*

Towards this direction, our schema provides two specific functionalities, *Attestation by Proof* and *Attestation by Quote* (see Figure 3.2), for enabling the automatic and secure establishment of trust between deployed platforms. This is part of the overall RAINBOW Attestation Toolkit: *For privacy, RAINBOW will leverage advanced crypto primitives, namely Direct Anonymous Attestation (DAA) (Chapter 4), whereas for security and operational assurance, it will enable the provision of Configuration Integrity Verification*.

The evidence of the integrity state of the configuration properties is authenticated by the attached TPM. Key features provided include the: (*i*) the possibility for low-level fine-grained tracing capability (Attestation by Quote), and (*ii*) the option for privacy-preserving attestation (Attestation by Proof). The former is a significant feature because, once a platform is compromised, it can be immediately retracted without affecting the entire service graph chain, thus, catering to efficient service management and flexible slicing [13]. The latter enables the integrity verification of a designated platform without conveying other platform's information (or unnecessary information of the underlying host) to a remote entity (acting as the $\mathcal{V}$rf), in case of a malicious $\mathcal{V}$rf being aware of which components the underlying host and other processes have. This is of paramount importance in emerging fog based environments, leveraging such advanced security capabilities to support safety-critical services with strict security, trust, and privacy requirements [29].

The offered secure attestation trust extensions allow to assess and preserve the integrity of the deployed platform's Trusted Computing Base (TCB), at load-time and during system execution, by leveraging the capabilities of TPMs, and reducing performance impact by minimizing the necessary interactions with the host trusted component. It supports complete, configurable attestation that acquires binary signature chains from different unique registers, enabling advanced tracing capabilities to localize areas of compromise. Both schemes rely on the platform to access a TPM with irreversible PCRs. The privacy-enhanced feature builds on the use of an Attestation Key within the TPM that can only execute a cryptographic operation if a set of PCRs is in particular (trusted) state, inferring the correctness of the component.

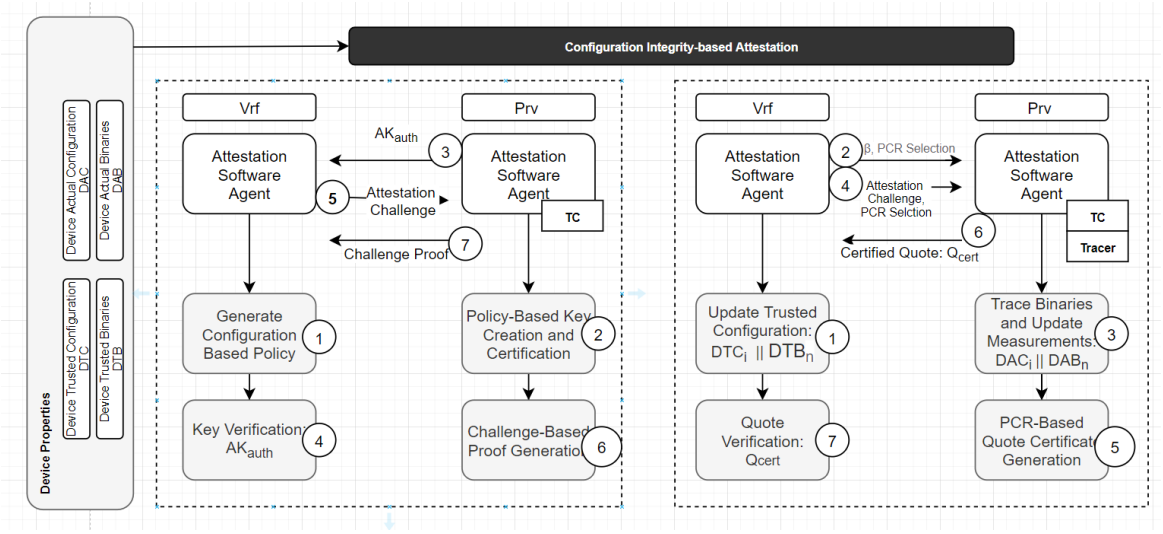

Figure 3.2: RAINBOW Trusted Extensions of platform Secure Remote Attestation: Attestation by Proof (**L**eft) and Attestation by Quote (**R**ight).

Figure 3.2 presents the information flow of the RAINBOW integrity verification capabilities between a Prover ($\mathcal{P}$rv) and a Verifier ($\mathcal{V}$rf): In a nutshell, this attestation toolkit detects offline and online attacks on mutable files (configuration properties) by verifying their hashed digest with a trusted reference measurement extracted from a corresponding PCR on the $\mathcal{O}$rc. Attestation reports produced by the (verified) platforms can include as much information as required based on the already defined attestation policies (including the configuration properties to be traced).

Attestation policies must be expressive and enforceable and can be dynamically updated by the $\mathcal{O}$rc. After defining proper policies, this engine can proceed to periodically (or on-demand) attest to the modeled configuration properties representing the current state of the target platform. A platform is trusted if its state (at that time) matches the (already measured) reference state. As each platform is a combination of multiple software processes running, its hashed digest defines its state. By comparing the hashed digest (at any given time) to the reference (expected) hashed digest of the platform, provided by the $\mathcal{O}$rc, we can determine the platform trustworthiness.

3.3 RAINBOW Zero-Touch Integrity Verification Building Blocks

The core of our schemes (Figure 3.2) is the manageability of mutable configurations throughout the lifespan of a fog node operation and is accomplished by having the $\mathcal{O}$rc mediating any security-critical updates towards the deployed platforms. Whenever the orchestrator extracts new security attestation policies (due to potentially identified new vulnerabilities), containing updates on the set of configuration properties to be verified, or due to changes in configurations, it proactively determines the update's expected implication by accumulating the artificial vPCR construct of the corresponding device (**Step 1R**). The $\mathcal{O}$rc requests the platform to similarly accumulate its PCRs to reflect potential changes (**Steps 2R-3R**). This update request contains only the PCR index i that must be updated and a configuration file identifier to measure. Upon receiving such update requests, the device then invokes the $\mathcal{T}$rce to measure the requested file(s) and subsequently invokes the attached TPM to extend PCR i with the new measurement. The simple update protocol is depicted in Figure 3.3.

In this context, to initiate the re-measurement process of a device, the $\mathcal{O}$rc sends an *Update*

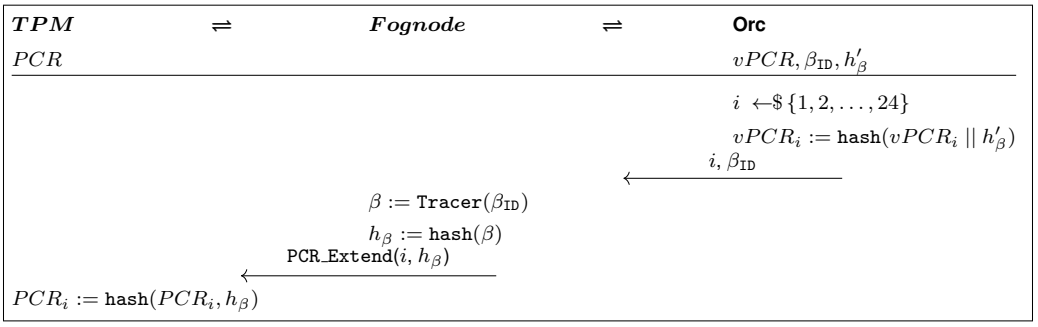

Figure 3.3: Update PCR Measurements

Measurements request detailing which file object should be re-measured (using $\mathcal{T}$rce) and into which PCR registers it should be registered. Note that the $\mathcal{O}$rc knows what the correct PCR values should be since it also accumulates the artificial PCR registers (vPCRs), as part of the attestation policy, corresponding to the target device. To perform a verifiable assessment of the current state of a device's PCRs, the $\mathcal{O}$rc sends an *Attestation by Quote* request detailing which PCR registers should be included in the quote, denoted $\mathcal{I}$, and a nonce n to enforce freshness and prevent replay attacks. After the device has securely instructed its TPM to construct the necessary quote certificate and signature (over the certificate), it forwards them to the $\mathcal{O}$rc. If the signature over the quote is deemed correct (signed by the device's EK_{priv}), the certificate can the be verified for determining whether it contains the "magic header" (proving that it was generated inside the TPM and whether the PCR values correspond to the trusted PCR values (vPCR) that were artificially accumulated on the $\mathcal{O}$rc when the $\mathcal{V}$F was last updated.

Furthermore, the privacy-preserving attestation (i.e., local attestation) requires the use of specialized signing keys, called attestation keys (AKs), which can be bound to specific PCR contents, hence making an AK operable *iff* the PCRs reflect the particular PCR state in which the AK is bound to. However, to retain the viability and correctness of such an attestation (despite mutable PCRs), we must create and bind a new attestation key whenever a device's configuration is updated. Since the key creation process is best achieved locally, the $\mathcal{O}$rc requests a device to create a new attestation key based on the trusted measurements that were artificially accumulated before requesting the device to update its PCRs. First, the $\mathcal{O}$rc computes an Extended Authorization (EA) policy digest based on the trusted measurements (**Step 1L**), denoted $h_{\texttt{Pol}}$, which reflects the trusted state in which the AK must be bound to. The policy digest is then deployed together with a subset of PCRs, $\mathcal{I}$, to which the policy applies. Upon receiving such a request, the device is responsible for creating the attestation key on the TPM (**Step 2L**). To trust that the policy is actually enforced and that the state of the device is conformant to the policy, the device must present indisputable evidence towards the: (*i*) creation of AK happened inside the TPM, (*ii*) provided policy digest governs the key, and (*iii*) proof originates from a distinct TPM.

Fig. 3.4 presents the underpinnings of the protocol for AK creation, where a device initiates the process by constructing a "key template" based on the received policy digest. This template dictates the fundamental properties of the key, i.e., whether it is a signing key, decryption key, or both, and whether it is restricted (operates *solely* on TPM-generated objects). The template is passed to the TPM, which creates an AK as a child key of EK. This process outputs a creation hash $h_{\texttt{Create}}$ and a ticket T, where T is computed with the inclusion of a secret value (`Proof`) known only by TPM, which proves that the TPM created the AK (**Step 2L**). The ticket is subsequently passed as an argument to the $certifyCreation$ functionality of the TPM, together with AK, to enable AK's certification using EK, which, due to being restricted, requires such indisputable evidence about the provenance of an object. The certificate and its signature are then

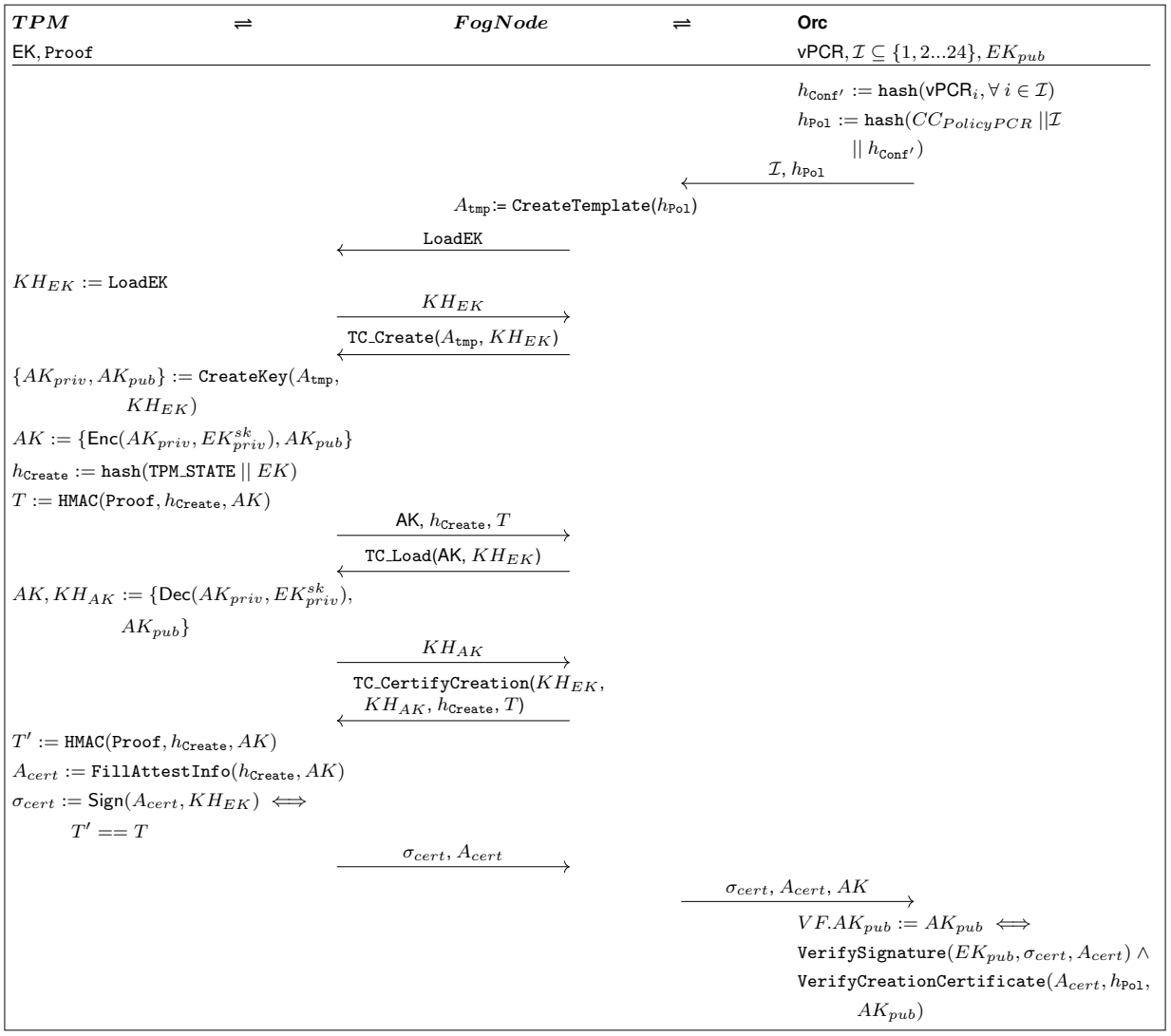

Figure 3.4: Create new Attestation Key

sent to the $\mathcal{O}$rc for verification (**Step 3L-4L**). The generated AK is trusted *iff* the signature over the certificate is verified to be authentic, based on the device's EK_{pub}. The certificate reflects that the AK was created to require the correct attestation policy to be used for signing operations and that the certificate includes a (public) value called the "magic header" whose presence proves that the signed object was created internally on TPM.

Attestation by Quote. The protocol for remote attestation using the TPM quote structure is presented in Fig. 3.5. In this protocol, the $\mathcal{O}$rc sends a nonce n (to enforce freshness and prevent replay attacks) and a selection of PCRs to attest, $\mathcal{I}$ (**Step 4R**). The device subsequently passes these arguments to the attached TPM which constructs a quote structure comprising the current values of the chosen PCRs, and signs it with its EK (**Step 5R**), which as with AK creation, proves that the quote structure is internal to the TPM. The quote certificate and signature are then sent to the $\mathcal{O}$rc (**Step 6R**). The quote and its signature are successfully verified by the engine (**Step 7R**) *iff* they are valid, and if the PCR values correspond to the artificial reference values (vPCR) managed by the $\mathcal{O}$rc.

Attestation by Proof. In the Attestation by Proof protocol (Fig. 3.6), the $\mathcal{O}$rc *only* sends a fresh nonce n to a device (**Step 5L**). If the device presents $Sign(n, AK_{priv})$ (**Step 6L**), where AK is a fresh and verified AK, then this is indisputable evidence that $\mathcal{V}$F is in a trusted state (**Step 7L**). Note, that both the Attestation by Proof and Quote can *only* attest to the last known

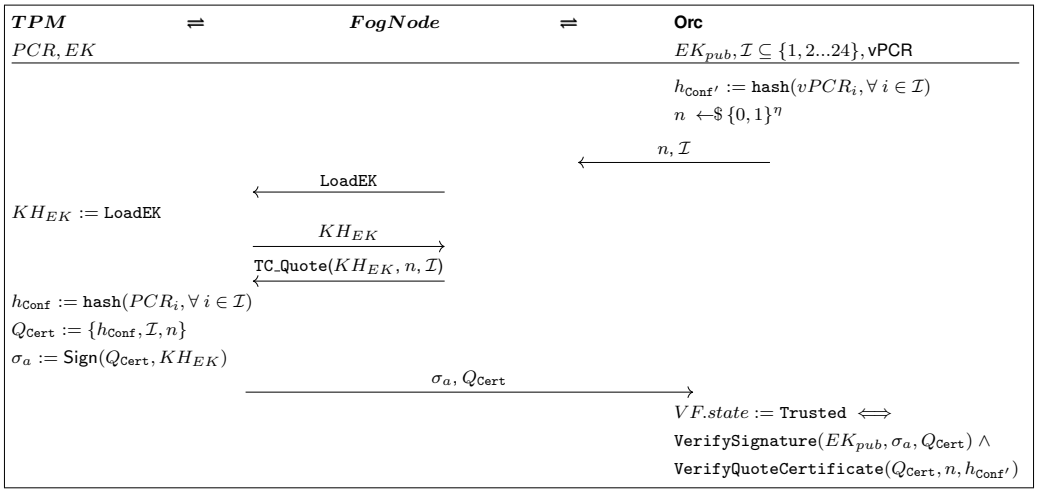

Figure 3.5: Attestation by Quote

measurement. Thus, both attestation schemes are tightly coupled to run in conjunction with the update of measurements protocol for achieving run-time device integrity.

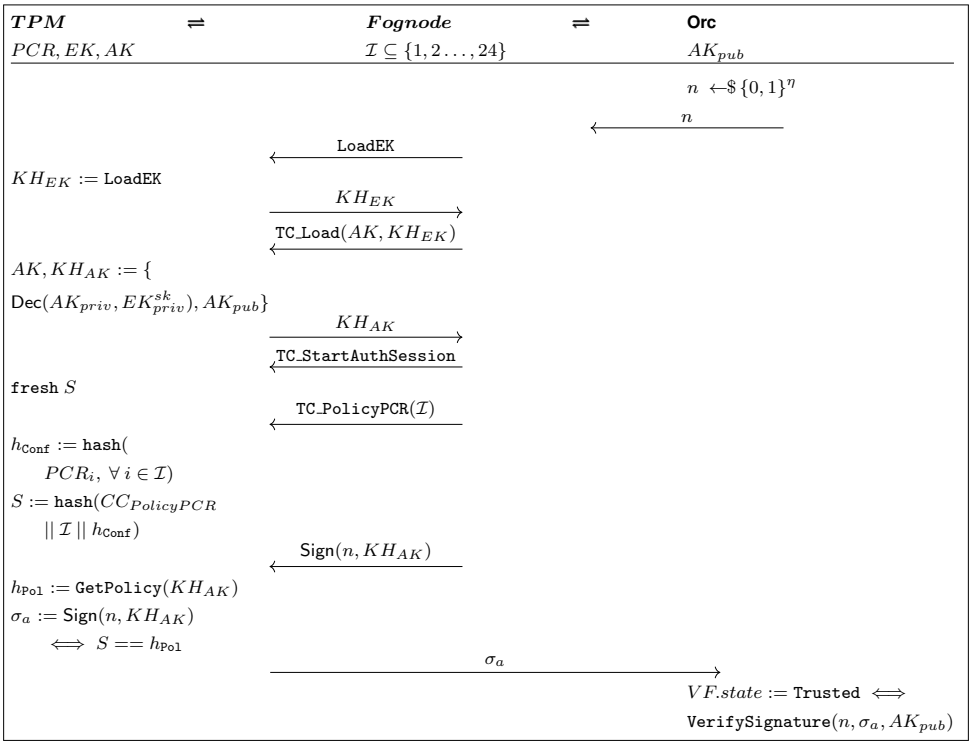

Figure 3.6: Attestation by Proof

3.4 Experimental Performance Evaluation

We have also proceed in a full implementation and evaluation of the RAINBOW set of secure remote attestation enablers in order to identify the overhead that this type of secure enrollment process can add on top of the underlying networking mechanism. While this experimentation is based on the benchmarking of both the *Attestation by Proof* and *Attestation by Quote* mechanisms, when running as standalone components, it provides us with a good starting point for the next integration activities of the overall RAINBOW trust overlay mesh network for delivering **the high-level functionalities related to secure (edge and mesh) device identification and**

Table 3.2: Timings of integrity verification protocols (time in ms). Note that the hashing is done without any secure hashing schemes and might be slower in practice.

Command	Activity	Mean	95% (low)	95% (high)	Description
CreateAK	Prepare	0.01	<0.01	0.01	Compute expected vPCR
	Create	15.92	15.80	16.05	Create AK in $\mathcal{TC}$
	Verify	1.03	1.01	1.05	Verify certificate and key
	Total	16.96	16.81	17.11	
Update	Prepare	<0.01	<0.01	<0.01	Extend vPCR
	Hash/Extend	1.42	1.35	1.49	Hash file(s) and extend PCR
	Total	1.42	1.35	1.49	
Quote	Prepare	0.02	0.01	0.03	Create a nonce
	Quote	8.67	8.56	8.78	Sign PCRs with EK
	Verify	0.83	0.80	0.85	Verify quote and certificate
	Total	9.51	9.37	9.65	
Proof	Prepare	0.01	<0.01	0.02	Create a nonce
	Sign	10.83	10.76	10.89	Sign nonce
	Verify	0.84	0.79	0.88	Verify signature
	Total	11.67	11.56	11.79	

integrity, data integrity and confidentiality, anonymity and resource integrity as described in the overall framework reference architecture (see Deliverable D1.2 [21]).

Experimental Setup. Our testbed is deployed on a computer equipped with an Intel(R) Core(TM) i7-8665U CPU @ 1.90-2.11GHz running the Windows 10 OS. The main goal of this setup is to evaluate the potential overhead of using a TPM that will, in turn, allow us to assess the overall protocol scalability towards providing verifiable integrity evidence. Therefore, we have opted out of creating a true scale test environment with separate entities, but a single binary file containing all components. To evaluate the performance of attestation trust extensions, we constructed the protocols and tested them against IBM's software TPM V1628 using the IBM TSS(Section 2.1.3). Each experiment (protocol) is performed 1,000 times. Note that since we rely on a software TPM as the RoT, of a device, we chose to create an attestation primary-key as an alternative to the EK for key storage, which adds a small overhead each time the AK is used. Also, we chose to use an ECC key as the EK, instead of an RSA-based EK. However, as long as the key is sequestered in the TPM, either approach is secure.

Timings are gathered by executing the experiments for 1000 runs, calculating the mean and standard deviation to acquire the 95^{th} percentile.

Performance Results. Our experiments (Table 3.2) highlight the efficiency of our protocols. The entire process of creating an AK takes no more than (approximately) 17 ms (on average), while including the update of binary measurements still requires less than 20 ms (see Section 3.4.1 for more details and a comparison to a HW-based TPM implementation). The enhanced attestation schemes are also efficient (< 12 ms), however, without considering any possible network delay that may be present when communicating the attestation data between the Prv and Vrf. With both supporting routines and attestation schemes being extremely lightweight, we can achieve low-cost, rapid attestation capabilities and provide advanced trust assurance services without consuming a lot of computational resources. Such capabilities not only ensure trust from the perspective of the entire service graph chain but further facilitate bilateral trust assurance (even) between different service graphs. In general, higher levels of trustworthiness result in more resources being needed. That is why it is imperative for the attestation protocols to be lightweight enough without, however, impeding on their accuracy and correctness. To better demonstrate the achieved effectiveness, we use Eq. 3.1 to determine how fast we can detect binary manipulation.

In the worst-case scenario, where an Adv tampers with a binary just after an update, she will remain undetected for *at most* 293.40 ms, if we utilize as little as 20% of the CPU time.

The ease of operating RAINBOW's attestation protocols, including their efficiency, makes the framework highly applicable to be integrated into large-scale networks. While - at this stage - we did not take processing- or network-delay into consideration and only use the AK once, the experiments show that the time of detection is in the order of seconds. In Figure 3.8, we further see that even with 10% utilization, we can still detect a change after ≈ 1 second, making it extremely difficult for an Adv to manipulate RAINBOW's attestation protocols.

3.4.1 Timings and Benchmarks

In the context of RAINBOW, we do not consider Adv that can perform transient attacks whereby alterations to binaries are only detectable for a short time. Thus, any alterations to binaries by an Adv will be detected when the device is re-measured and attested, as shown in Figure 3.7. The *advantage* of an Adv is defined as the time that she can remain undetected. If, for instance, the *Update of Measurements* and *Attestation by Quote* protocols are executed immediately after the attack, then we will be able to detect any incompliant configurations from the quote structure. However, *Attestation by Proof* will inevitably take longer to complete since the creation of a new AK must occur in-between the *Update of Measurements* and *Attestation by Proof* protocols. Let t_d denote the time of detection (hence, a large t_d is desirable to Adv), u the time to execute the update routine, c the time to execute the creation of a new AK, a the time to execute the attestation routine, and n the number of device's that, in consecutive order, conduct *Attestation by Proof* on a specific device using a shared and verified AK_{pub} of that device, respectively. We further use the variable t_{CPU} to specify the amount of CPU resources allocated to execute these routines, e.g., for 20% utilization we have $t_{CPU} = 0.20$. Using Eq. 3.1 we calculate the time until the Adv is detected (t_d) (Figure 3.8). As we can see, the detection time (t_d) increases linearly with n (a) but decreases as we allocate more resources, t_{cpu} (b).

$$t_d = \frac{2c + a(1+n) + u}{t_{CPU}} \tag{3.1}$$

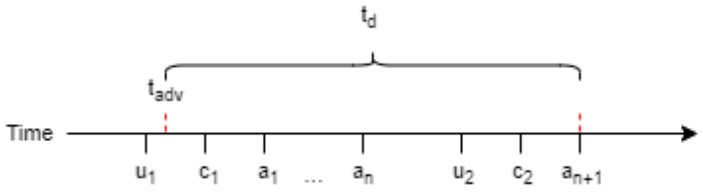

Figure 3.7: Visual representation of how long an $\mathcal{A}$dv can go undetected.

Implementation Note. Writing protocols in terms of TPM calls requires reading and understanding the TPM 2.0 specification and this makes TPM development challenging and causes a high-barrier of entry. While the TPM 2.0 specification was designed to be easily maintainable, it is nevertheless challenging to read mainly due to its sheer size. It consists of over 1400 pages split into four parts which not only cover the core specifications, but also numerous errata covering the continuous development of the TPM specification. Therefore, a particular TPM will be based on the core specification and all of the relevant errata which it implements.

HW-based TPM Timings. The timings for executing the individual TPM commands of the presented attestation protocols are presented in Tables 3.3 and 3.4, and are performed using IBM's

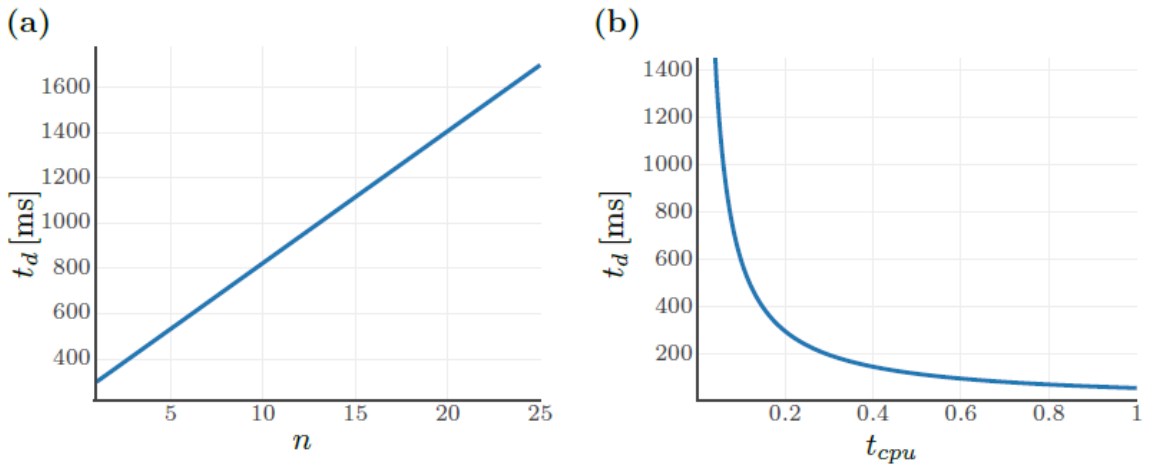

Figure 3.8: (a) Changing the number of attestations each key has to do and its impact on the time of detection (20% utilization) and (b) shows how different utilization of resources impact the time of detection with one AK use.

Table 3.3: Mean time (in ms) of using SW- and HW-TPM for updating measurements and creating a new AK.

Command	SW	HW
Update		
TPM2_PCR_Extend	0.44	6.09
Total	0.44	6.09
Create		
TPM2_CreatePrimary	0.92	238.35
TPM2_Create	0.98	243.75
TPM2_Load	0.44	58.04
TPM2_Load	0.33	59.83
TPM2_CertifyCreation	5.18	123.13
TPM2_FlushContext	1.64	4.10
TPM2_FlushContext	1.60	3.69
TPM2_FlushContext	2.21	4.00
Total	13.3	734.89

Table 3.4: Mean time (in ms) of using SW- and HW-TPM for Attestation by Quote and Proof.

Command	SW	HW
Quote		
TPM2_CreatePrimary	3.36	244.96
TPM2_Load	1.57	51.69
TPM2_Quote	2.16	112.71
TPM2_FlushContext	0.93	3.71
TPM2_FlushContext	0.89	3.77
Total	8.91	416.84
Proof		
TPM2_CreatePrimary	3.33	241.38
TPM2_Load	1.70	54.17
TPM2_StartAuthSession	1.38	6.72
TPM2_PolicyPCR	0.37	10.71
TPM2_Sign	3.06	95.36
TPM2_FlushContext	0.97	4.12
TPM2_FlushContext	0.91	4.98
Total	11.72	417.44

software (SW) TPM V1628 and the Infineon (HW) TPM 2.0 chip. The mean time is calculated from repeating all experiments 1,000 times for the SW-TPM and 100 times for the HW-TPM. The values reflect the time between executing a command in the IBM TSS [**?**] V1.5.0 and until receiving a response. The SW-TPM timings are much faster than those when using the HW-TPM. Even though a hardware TPM has some degree of hardware accelerated cryptography, it still cannot measure itself with a modern CPU, and is not designed to do so. Applying Eq. (3.1) on the HW-TPM yields a time of detection as $t_d = (2 \cdot 734.89 \text{ ms} + 417.49 \text{ ms}\,(1+1) + 6.09 \text{ ms})/0.20 = 11.5$ s, which is indeed larger than that for the SW-TPM. However, these values are somewhat misleading since the host CPU's utilization does not have any effect on the HW-TPM as it executes the operations on the hardware chip itself. By evaluating the TPM command execution and ap-

plication timings, we can see that the most time-consuming operations are those executed on the TPM, which is why the impact of the CPU utilization using an HW-TPM is significantly lower. Removing this constraint from Eq. (3.1) gives us $t_d = 2.310$ s (excluding the host times, such as verification, nonce generation, etc.). Additionally, the “create primary key” function is extremely time-consuming, which is why it might be useful to load this AK from NV storage.

Chapter 4

Privacy-aware Service Graph Chains using RAINBOW Direct Anonymous Attestation

In order to provide novel implementations for fog-based environments, many challenges have to be overcome with **security** and **privacy** being critical pillars; especially in the context of safety applications where critical decisions are based on information collected by devices regarding their status or surrounding events (e.g., Urban Mobility Use Case [24]).

Therefore, seeking to design successful secure and privacy-preserving fog-oriented architectures, comprising of thousands of autonomous and intelligent fog and edge nodes, besides **operational assurance** - one has to cater for a number of properties like ***anonymity, pseudonymity, unlinkability, and unobservability*** and the strict trust requirements of a wide variety of multi vendor devices and platforms. The security, interoperability and connectivity in a dynamic network of fog nodes, gateways, services and applications across operations technology and information technology stakeholders requires strategic rethinking of policies and processes in the context of cyber-security, privacy and trust establishment.

Towards this direction, in RAINBOW, we will employ advanced cryptographic primitives (namely **Direct Anonymous Attestation** [19]) together with trusted computing techniques for facilitating the strict privacy considerations encountered in a variety of emerging applications. For instance, in the context of the Urban Mobility scenario - envisioned in RAINBOW - privacy is a key concern since the involved vehicle transmissions can be used to infringe the user's location privacy [52]. Many V2X applications rely on continuous and detailed location information, which if misused (all exchanged messages can be eavesdropped within radio range) can lead to the extraction of detailed location profiles of vehicles and path tracking [30]. Since there is usually a strong correlation between a vehicle and its owner [33], location traces of vehicles have the potential to reveal the movement and activities of their drivers. Two of the most prominent types of messages that are exchanged in the context of V2X are known as Cooperative Awareness Messages (CAM) and Decentralised Environmental Notification Messages (DENM) [28].

In the remaining of this chapter, we provide a detailed documentation of the enhanced DAA protocol that has been designed and implemented for providing strong privacy guarantees throughout the entire operation of a fog-based ecosystem. Details on the **basic building blocks, mode of operation and workflow of actions** are provided as well as a **comprehensive mapping of the TPM commands** that need to be securely executed by the underlying TPM. While in its current phase, this DAA block is positioned as a standalone component in the overall RAINBOW architecture [21], the end-goal is the integration of this enhanced DAA mechanism in the underlying RAINBOW routing mechanism towards the provision of a **secure overlay mesh network for**

delivering the high-level functionalities related to secure (edge and mesh) device identification and integrity, data integrity and confidentiality, anonymity and resource integrity

4.1 The Need for "Privacy-by-Design" In Fog-based Ecosystems

Privacy requirements have been well documented in the European Telecommunications Standards Institute (ETSI) TS 102 941 [27], and the OpenFog Consortium standards highlighting the following properties:

- *Anonymity:* ability of a fog/edge node to use a resource or service without disclosing its identity.
- *Pseudonymity:* ability of a fog/edge node to use a resource or service without disclosing its identity while still being accountable for that action.
- *Unlinkability:* ability of a fog/edge node to make multiple uses of resources or services without others being able to link them together (i.e., infer mobility patterns).
- *Unobservability:* ability of a fog/edge node to use a resource or service without others, especially third parties, being able to observe that the resource or service is being used.

In this context, **the actual identity of the sender is not required for ensuring the trustworthiness of a transmitted message**. It rather suffices to verify the **origin correctness**; a message has been sent by a valid "fog participant". Indeed, since exchanged messages might contain sensitive data, what is required is that certificates should not contain any identifying information that could *trace* them back to a particular device or platform. In an attempt to address this challenge, intensive efforts in academia and industry, led to the proposal of **PKI-based solution**s [31] **with privacy-friendly authentication services** through the use of short-term anonymous credentials, i.e., **pseudonyms**. *The common denominator in such architectures is the existence of trusted (centralized) infrastructure entities for the support of services such as authenticated node registration, pseudonym provision, revocation, etc.*

While it has been proven the security guarantees provided in such architectures, there are a number of challenges inherent to PKIs when it comes to **privacy**, **scalability**, and **operational assurance** [29]. In its core, the possibility of security breaches has the potential to seriously weaken the technical security protection measures of PKIs, since in their current version the assumption is on the existence of a number of centralized trusted entities. However collusion or security incidents affecting certification authorities have grown more frequent in the recent past [26], **so the existence of a PKI architecture does not guarantee per se the enactment of trust between the peers and additional measures are necessary to reinforce a scalable community of trust** [9].

Therefore, **what is needed is to provide efficient, reliable and in timely and privacy-preserving communications to all fog nodes and their embedded TPMs**. The reliance on infrastructure entities within the overall architecture for such services raises questions towards a system's availability and scalability in the case of a technical fault or attack. In this context, **RAINBOW leverages anonymous credentials through the use of Direct Anonymous Attestation (DAA) addressing all the aforementioned limitations, i.e., privacy, security, and accountability.**

One of the biggest advantages of RAINBOW DAA scheme is its **decentralized nature** resulting in shifting the trust from the back end infrastructure to the fog/edge nodes. Applying the DAA protocols results in the redundancy (and removal) of the PKI-based architecture: *fog nodes can now create their own pseudonyms, and DAA signatures are used to self-certify each such credential that is verifiable by all verifiers*. Furthermore, nodes have total control over their privacy, as no trusted third-party is involved in the pseudonym creation phase. This means that it is infeasible for any third-party to reveal the identity of another device assuring that pseudonym resolution is not possible in our solution.

4.2 Direct Anonymous Attestation Building Blocks

DAA [16] is a platform authentication mechanism that enables the **provision of privacy-preserving and accountable authentication services**. DAA is based on group signatures that give strong anonymity guarantees. The key security and privacy properties of DAA are:

- *User-controlled anonymity*: Identity of user cannot be revealed from the signature.
- *User-controlled linkability*: User controls whether signatures can be linked.
- *Non-frameability*: Adversaries cannot produce signatures originating from a valid trusted component.
- *Correctness*: Valid signatures are verifiable, and linkable, where needed.

A DAA scheme considers a set of Issuers, hosts, Trusted Components (TCs - TPMs in the context of RAINBOW), and verifiers (Figure 4.2); the host and TC together form a trusted fog node. The Issuer is a trusted third-party responsible for attesting and authorizing platforms to join the network (through the execution of the previously described zero-touch configuration integrity verification process - Chapter 3). *In the context of RAINBOW, the role of this entity is undertaken by the Orchestrator ($\mathcal{O}$rc).* A verifier is any other system entity or trusted third-party that can verify a platforms' credentials in a privacy-preserving manner using DAA algorithms; without the need of knowing the platform's identity. The Elliptic-curve cryptography (ECC) based DAA is comprised of five algorithms SETUP, JOIN, SIGN, VERIFY and LINK.

- SETUP - The system parameters must be chosen and the Issuer needs to generate its keys. The system parameters and the Issuer's public keys are then published and available to the cluster and to anyone who needs to verify the validity of a signature.
- JOIN - A Host using a TC joins the group and obtains an Attestation Key Credential (AKC) for an ECC-DAA key created by the TC. The key can then be used to anonymously sign a message, or attest to data from this TC.
- SIGN - Using the ECC-DAA key, for a range of signing operations.
- VERIFY - Verifying a signature and returning true (valid) or false (invalid).
- LINK - Checking two signatures to see if they are linked and returning true (linked) or false (un-linked).

t	a security parameter.
bsn	Linking base, either a special symbol, $\perp$, or an arbitrary string.
p	A large prime number.
n	A prime number.
$\mathbb{G}_1$	An additive cyclic group of order n over an elliptic curve. The curve has points with co-ordinates in $F_p \times F_p$.
$\mathbb{G}_2$	An additive cyclic group of order n over an elliptic curve.The curve has points with co-ordinates in $F_{p^2} \times F_{p^2}$.
$\mathbb{G}_T$	A multiplicative cyclic group of order n.
$\hat{h}$	A bilinear map function $\hat{h} : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ such that for all $P \in \mathbb{G}_1, Q \in \mathbb{G}_2$, and all positive integers a, b, the equation $\hat{h}([a]P, [b]Q) = \hat{h}(P, Q)^{ab}$ holds. This function is also called a pairing function. Note – this was called e in the ISO documents.
H	A cryptographic hash-function.
m	Message to be signed.
O_E	The elliptic curve point at infinity.
P_1	Generator of $\mathbb{G}_1$.
P_2	Generator of $\mathbb{G}_2$.
Z_p*	The multiplicative group of invertible elements in Z_p.
Z_p	The set of integers in $[0, p-1]$.
Z_p*	The set of integers in $[1, p-1]$.
$[k]P$	Multiplication operation that takes a positive integer k and a point P on the elliptic curve E as input and produces as output another point Q on the curve E, where $Q = [k]P = P + P + \ldots + P$, i.e., the sum of k copies of P. The operation satisfies $[0]P = [O_E]$ and $[-k]P = [k](-P)$.
$[x, y]$	The set of integers from x to y inclusive, if x, y are integers satisfying $x \leq y$.
$\{0,1\}^*$:	$str, \mathcal{C}, \hat{\mathcal{C}}, \bar{s_2}, \mathcal{A}, m$.
$H : \{0,1\}^* \rightarrow \mathbb{Z}_p$:	H_1.
$H : \{0,1\}^* \rightarrow \mathbb{Z}_n$:	H_4, H_5, H_9, H_{10}.
$H : \{0,1\}^* \rightarrow \mathbb{Z}_{2^{256}}$:	$H_2, H_3, H_6, H_7, H_8, H_{11}, H_{12}, H_{13}H_{14}$.
Messages:	m.
Signatures:	$\sigma_{ch}, \sigma_{ps}, \sigma_{ra}$.
$X \parallel Y$	is used to mean the result of the concatenation of data items X and Y in the order specified. In cases where the result of concatenating two or more data items is signed as part of one of the mechanisms specified in this part of ISO/IEC 20008, this result shall be composed so that it can be uniquely resolved into its constituent data strings, i.e. so that there is no possibility of ambiguity in interpretation. This latter property could be achieved in a variety of different ways, depending on the application. For example, it could be guaranteed by (a) fixing the length of each of the substrings throughout the domain of use of the mechanism, or (b) encoding the sequence of concatenated strings using a method that guarantees unique decoding, e.g. using the distinguished encoding rules defined in ISO/IEC 8825-1.

Figure 4.1: Notation Used

A DAA scheme enables a signer to prove the possession of the issued credential C to a verifier by providing a signature, which allows the verifier to authenticate the signer without revealing the credential C and signer's identity. In a nutshell, DAA is essentially a two-step process where, firstly, the registration of a fog node executes and during this phase the device chooses a secret key (SETUP). This secret key is stored in secure storage so that the host cannot have access to it. Next the device talks to the issuer so that it can provide the necessary guarantees of its validity (JOIN). The issuer then places a signature on the public key, producing the Attestation Identity Credential (AIC) cre. The second step is to use this cre for anonymous attestations on the platform (SIGN), using Zero-Knowledge Proofs [32]. These proofs convince a verifier that a message is signed by some key that was certified by the issuer, without knowledge of the TC's DAA key or cre (VERIFY).

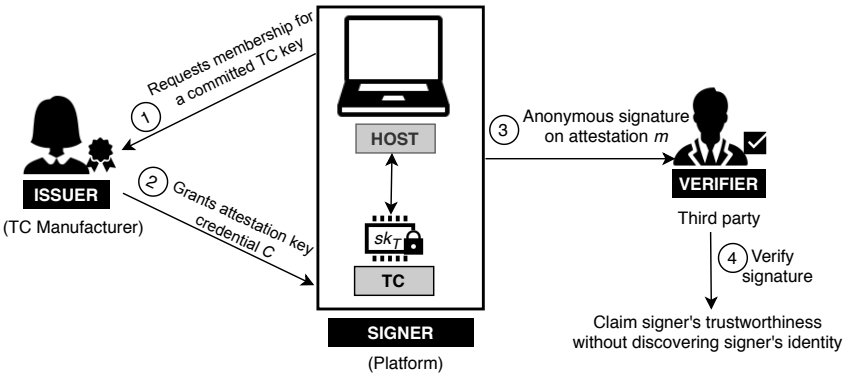

Figure 4.2: An overview of the entities involved in a DAA protocol

4.3 RAINBOW DAA Scheme

In our RAINBOW DAA scheme, it is only the $\mathcal{O}$rc that we assume as trusted; as aforementioned, **the orchestrator is responsible for authenticating fog nodes through the JOIN protocol**. In our context, **fog nodes are the combination of a *host*, that is the normal computing platform "normal world", and a TPM that executes in the "secure world"**; together they form the device which we refer to from this point onwards as the fog node. We also have an additional role - this of *verifiers* which are other fog nodes or third-party service, etc.

We have to highlight that our proposed solution assumes on-board TCs that support the functionalities described in Chapter 2: (*i*) *isolation*: separate and protected from the host in the event of compromise, (*ii*) *protected execution*: ensures the operation is executed and not interfered with, and (*ii*) *secure storage*: storage which is only accessible by the TC if the platform is in a "good" state.

Figure 4.3 defines the implementation of our RAINBOW DAA protocols. We describe each protocol execution by defining the responsibility of all system actors and separate the roles of the TPM and host. This allows us to better reason against the required functionality of a TPM. The reader is referred to Table 4.1 for fully expanded explanations of the notations contained below. *This is a high-level conceptual description of the workflow of actions - more details on the crypto operations been performed and the TPM functionalities/commands leveraged are ascribed in Section 4.4.1.*

4.3.1 Fog Node Registration

The first step for a node acquiring its certificates (after the correct execution of the Configuration Integrity Verification, described in Chapter 3, towards the secure enrollment of only those nodes that are at a "correct state") consists of two phases: SETUPfor generation of keys and the enrollment phase to the $\mathcal{O}$rc (JOIN). We assume that during manufacture time, the TPM will have a unique `DAASeed` installed, a non-monotonic counter cnt, and the hardware will be endorsed by the manufacturer through means of burning the endorsement key pair: $sk_{ek_{tc}}$ / $pk_{ek_{tc}}$ into the TPM. For the SETUPphase the $\mathcal{O}$rc publishes its public key pk_I and the security parameters K_I. A node's TPM generates a DAA key pair: sk_{tc} / pk_{tc} using K_I, and publishes its public key pk_{tc}. The TPM then releases the public keys $pk_{ek_{tc}}$ and pk_{tc} to the fog node.

The details of the JOINprotocol are shown in Figure 4.3. By the end of the protocol the newly registered platform will have acquired a VID Certificate (cre) certifying that the node has a valid TPM which has been enrolled with the $\mathcal{O}$rc. To initiate the JOINprotocol, a node sends the $\mathcal{O}$rc

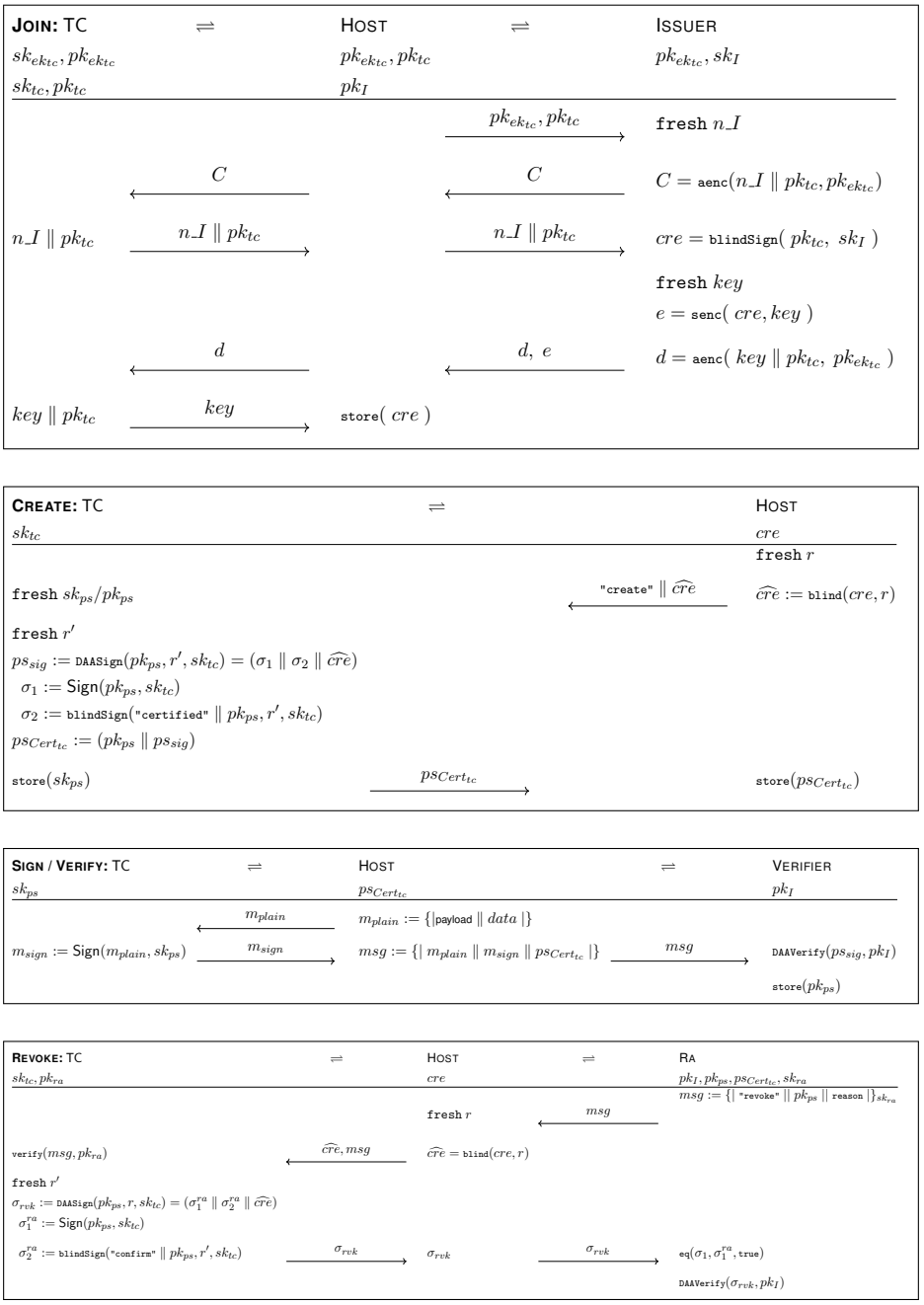

Figure 4.3: High-level Overview of the RAINBOW DAA Protocol Interfaces.

its public key indicating it wants to join the network (Step 1). The $\mathcal{O}$rc responds to the vehicle with a fresh challenge C which only the valid TPM can open. The node then forwards C to its TPM via a secure I/O (Step 2). The TPM opens the challenge, confirms its validity, and sends the response to the host node, which in turn, responds to the $\mathcal{O}$rc with the recovered data items (Step 3). The $\mathcal{O}$rc verifies the received response, confirming that the vehicle possesses a valid TPM. Following this verification, the $\mathcal{O}$rc creates the credential cre, and a fresh symmetric session key. The credential cre, encrypted with the session key, is sent to the node, along with an encryption of key intended for the TPM, as e and d respectively in Step 4. Finally, the node uses the TPM to decrypt d, recovering the key. The TPM verifies the validity of d and then releases key to the node (step 5). The node can then decrypt e (using key) to recover the certificate cre. Finally, it

verifies cre using pk_I and stores it for future use.

By the end of this protocol, if successful, the node is an authenticated and legitimate member of the fog cluster, and ready to register to any of the provided services including the underlying mesh networking stack (Chapter 6).

4.3.2 Anonymous Credentials Creation

The creation of pseudonyms (CREATEin Figure 4.3) lies within the fog nodes, allowing the shift of trust from a third party to locally within the end-points. This is made possible by all nodes being equipped with a TPM, that is responsible for generating the pseudonyms in an environment that enables protected execution, isolation and secure storage.

Creating new pseudonyms for a nodes does not require any external network communication, and all message exchanges in the CREATEprotocol take place over secure I/O between the host and TPM. To initiate the creation process the host blinds the cre with freshly generated random nonces, and sends a "create" request to the TPM with $\widehat{cre}$. Alternatively, the node can choose not to "blind" its credential and create pseudonyms which are linkable. While this is bad practice, it does demonstrate that anonymity, pseudonymity, unlinkability and unobservability are under the control of the fog node (based on the policies already circulated by the $\mathcal{O}$rc). Upon receipt of the pseudonym creation request, the TPM creates a fresh pseudonym key pair sk_{ps}/pk_{ps} and fresh random r. Using the `DAASign` algorithm the TPM creates two signatures: σ_1 - the public pseudonym key signed with the DAA secret key sk_{tc}, and σ_2 - a blind signature of the certified pk_{ps} key; ensuring the generated pseudonyms are not linkable. σ_1 is a "link token" which is created for the purpose of revocation, discussed in Section 4.3.4. Once the pseudonym signature is produced, ps_{sig}, the pseudonym certificate, $ps_{Cert_{tc}}$, is produced that is constructed from the public pseudonym key pk_{ps} and the pseudonym signature. The TPM concludes by storing the generated pseudonym secret key sk_{ps} and returns the pseudonym certificate to the host for use in networking communication.

By the end of this protocol a fog node can use its pseudonyms to communicate with the various services, such that the use of services are anonymous, unlinkable and unobserverable; whilst still being held accountable for its use of the services.

4.3.3 Network Communication

Through the use of DAA SIGN / VERIFYphases, secure and privacy-preserving exchange of messages are achieved by using the already generated pseudonyms. The following protocol details how authenticated message exchanges between fog/edge nodes.

To initiate a communication, the node maye creates a message that wants to either broadcast or unicast to other system nodes (Chapter 6). In our example, the node creates a plain unsigned message, m_{plain}, including binary data information. The TPM is given m_{plain} and signs it using the current pseudonym secret key, and responds to the node with a valid signature for m_{plain}. The node then constructs the complete message, msg, to broadcast (or unicast) to its surrounding nodes belonging to the same cluster. msg is constructed from the plain message, the message signature and the current pseudonym certificate $ps_{Cert_{tc}}$. The surrounding nodes (VERIFIER) receive msg, and first verify that the contained $ps_{Cert_{tc}}$ was created by a valid TPM that has been authorised by the $\mathcal{O}$rc. To achieve this the verifying node extracts ps_{sig} from the received $ps_{Cert_{tc}}$, and uses `DAAVerify` and the $\mathcal{O}$rc's public key pk_I to confirm the pseudonym was created by a valid TPM.

4.3.4 Revocation

As aforementioned, one of the most critical services in a fog-based ecosystem is **revocation**. In our protocol (REVOKEin Figure 4.3), we demonstrate how this is achieved, using DAA, whilst preserving privacy, and confirmation that the node was revoked. Revocation messages have linkable signatures to guarantee the correct reception of a revocation command by the node in question. Prior to the execution of this protocol, we assume a number of reports containing a misbehaving node's pseudonym have been issued, and the decision to revoke the node has been made based on strong evidence.

The respective authority (can also be the $\mathcal{O}$rc) initiates the REVOKEprotocol by creating a signed revocation message msg using its secret key sk_{ra}. It broadcasts msg containing the public pseudonym key, pk_{ps}, that needs to be revoked. All nodes receive the revocation message since the hosts are required to forward them to their TPMs, and furthermore they generate fresh random nonces and blind the credential producing $\widehat{cre}$ which is again forwarded to their TPMs. The TPM recognises this message as a revocation request, and verifies that the pseudonym public key was generated by the TPM and prepares to respond to the authority. The TPM generates some fresh random r, and uses the `DAASign` algorithm to produce the revocation confirmation signatures σ_1^{ra} and σ_2^{ra}. σ_1^{ra} is a deterministic signature that is linkable with σ_1 confirming the revocation is designated for this node. Then, σ_2^{ra} is a signed commitment to confirm that the pseudonym was revoked. As a consequence of σ_{ra} being produced, the TPM deletes all pseudonyms and its DAA key pair sk_{tc}/pk_{tc}. The TPM responds to the node with the revocation confirmation σ_{rvk}, which is then sent to the authority. Upon reception of the revocation confirmation, the authority verifies that σ_1^{ra} is the same signature as σ_1 from the pseudonym certificate implying that the correct node has revoked itself. The entire signature σ_{rvk} can be verified using `DAAVerify` as being signed by the TPM that belongs to the misbehaving node.
By the end of this protocol, there are strong guarantees that the node in question has been revoked without the need of any pseudonym resolution. The authority has verifiable evidence, from the node, that it has performed the revocation enforced by the TPM. In the event of a node revocation, it has to re-run the JOINprotocol before being able to re-join the fog cluster and acquire new credentials.

4.4 TPM Commands Instantiation & State Diagrams

Based on the above description of the DAA phases and protocols, the main goal of integrating such an advanced privacy-preserving mechanism in RAINBOW is for the: (i) data transfer preserving **anonymity** and **privacy**, using the **Direct Anonymous Attestation (DAA)** functionality, (ii) attestation of the correct state of fog nodes (on demand) during run-time, and (iii) the establishment of a secure and anonymous communication channel between the fog nodes themselves or with the $\mathcal{O}$rc. Thus, the derived trust model captures the necessary requirements for the correct execution of these services:

R1. Correctness: Valid DAA signatures are verifiable, and linkable, where needed. This also requires the correct execution of the protocol even in the presence of an adversary having compromised part of the host $PLATFORM$. For instance, only valid and trustworthy TPMs can join the system by ensuring that the endorsed TPM keys have not been previously compromised;

R2. User-controlled anonymity: Identity of the device cannot be revealed from the DAA signature. This means that an adversary who does not know the $PLATFORM's$ private key cannot link a signed message to the TPM of this platform;

R3. User-controlled linkability: Device control whether signatures can be linked. A device has control over its DAA credential and can decide whether or not to "blind" it through the use of a single or different basenames bsn;

R4. Non-frameability: Adversaries cannot produce signatures originating from a valid trusted component.

Figure 4.4 depicts the state diagrams for the aforementioned functionalities (further elaborated in Section 4.4.1): The first one denotes the correct execution of the core DAA phases, namely the SETUP and JOIN phases (upper branch), for certifying the TPM used by a host platform from the $\mathcal{O}$rc, and the SIGN (or VERIFY) phases (lower branch) for signing/verifying message digests, m_i, originating from the fog nodes. The second diagram denotes the establishment of a secure communication channel between multiple fog nodes capturing the current TLS key establishment (e.g., to be replaced by an anonymous ephemeral key establishment scheme that will be designed in the context of RAINBOW - Section 5) that can be anonymously signed by the host $PLATFORM$ using its DAA key. Finally, the third one denotes the attestation and verification of the correct state of a fog node. Overall, the light green colour represents a state of the host $PLATFORM$ (and not the TPM itself), the light blue colour represents a state of the $\mathcal{O}$rc and the light grey colour represents a state of the TPM.

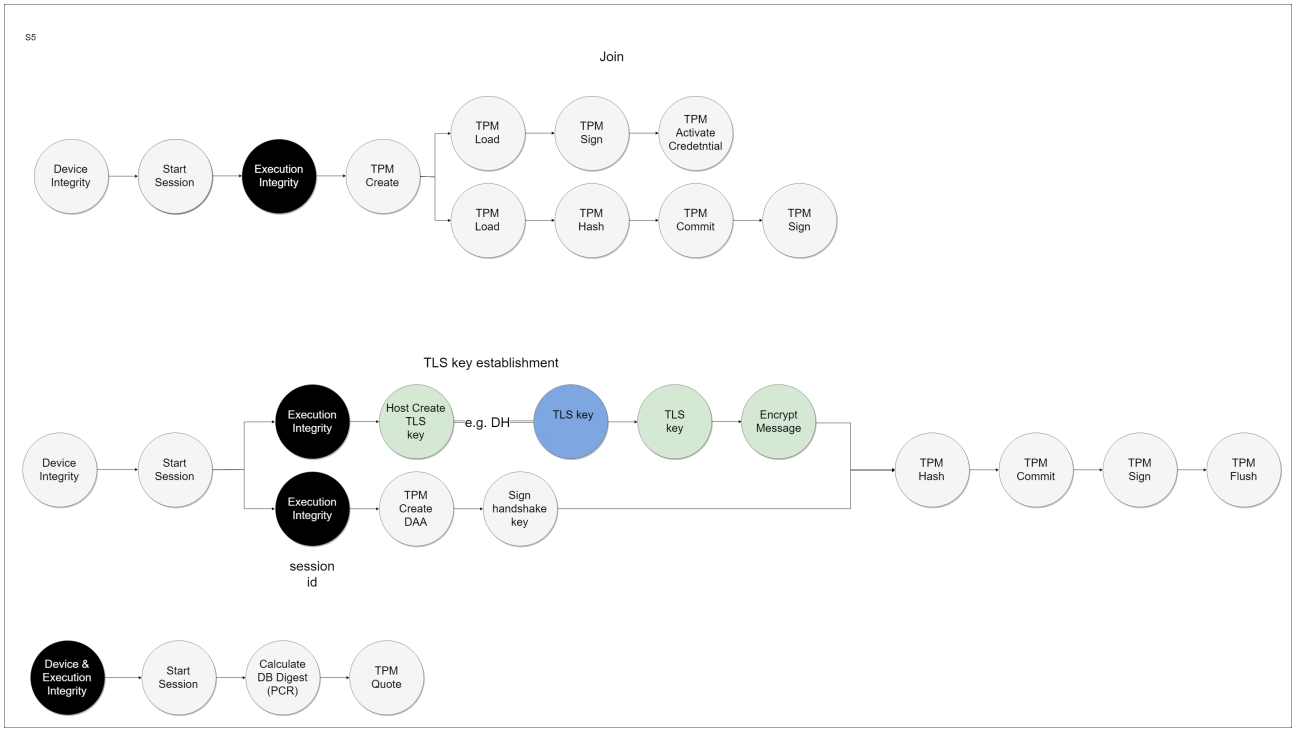

Figure 4.4: RAINBOW DAA State Diagrams

State Diagram of the DAA Protocol Execution (Figure 4.4 (a)):

Device Integrity: Integrity calculations of the applications installed on the TPM-equipped $PLATFORM$. Applications need to be attested against a whitelist of allowed and trusted application in-

stances (Chapter 3). The $\mathcal{O}$rc will be responsible for managing such whitelists in order to determine whether the DAA key generated by the TPM of a $PLATFORM$ is associated to a good software configuration. This property will be achieved by the TPM that can securely store the current system state in its Platform Configuration Registers (PCRs) and it will allow certain crypto operations to be performed with the DAA key only if the current state is the same as when the key was created.

Start Session: $PLATFORM$ applications to start a session with the TPM in order to create the DAA key and activate its credentials (SETUP and JOIN phases) and then execute a range of signing/verification operations: **SETUP**: The system parameters must be chosen and the $ISSUER$ needs to generate its keys. These parameters and the $ISSUER'S$ public keys are then published and available to anyone who wants to verify the validity of a signature. **JOIN:** a user $PLATFORM$ using a TPM obtains an Attestation Key Credential (from the $ISSUER$) for the DAA key create by the TPM.

Execution Integrity: Integrity of the execution of the TPM command flow by monitoring the Trusted Software Stack (TSS); i.e., invocations made by what applications, command configurations, parameters passed, etc. It reflects the continuous monitoring and attestation of the low-level system and behavioral properties to be performed by RAINBOW secure remote attestation enablers.

TPM Create: Creates a restricted key blob, in order to create the DAA key.

Activate Attestation Key Credential in JOIN Phase:

TPM Load: The TPM loads the created $ECC - DAA$ key. This key must be fixed to this TPM, fixed to this $TPM's$ key hierarchy and restricted to sign only message digests been created by itself (once data have been forward by the use $PLATFORM$ hosting the TPM).

TPM Activate Credential: Enables the association of a credential with an object (provided by the $ISSUER$) in a way that ensures that the TPM has validates the provided system parameters. In a nutshell, this TPM call is used to convince the $ISSUER$ that the $ECC - DAA$ key that it has received, has been generated by a TPM whose endorsement key has already been checked.

TLS key Establishment: Establishing a secure communication Channel from the $\mathcal{O}$rc to the $PLATFORM$.

TPM Sign: In the context of the JOIN protocol and in order to successfully complete the TPM activate credential command, it is necessary to perform one TPM sign based on the created DAA key.

The SIGN Operation:

TPM Load: The TPM loads the required keys for the signing operation

TPM Hash: Compute hash digests for the data bunches produced by the host $PLATFORM$ after being forwarded to the internal TPM. This operation provides the necessary guarantees that the message digest to be later signed have been created by the TPM itself. The results of the hash will be used in the signing operation that uses the restricted DAA key and the ticket returned by this command can indicate that the hash is safe to sign.

TPM Commit: After the attestation key certificate has been randomised, this command is used for preparing the parameters for the subsequent signing operation. It basically provides the required anonymity level by using either different or the same basename for the signing.

TPM Sign: After the execution of all previous states, the $DAAkey$ can now be used for any signing operation.

4.4.1 TPM Commands Mapping to RAINBOW DAA

In what follows, we put forth the detailed mapping of the underlying TPM commands (Section 2.1.3) that need to be executed per DAA phase and workflow of actions as described in Section 4.3. *This provides a detailed overview of the DAA implementation that has been performed in the context of RAINBOW and will be integrated in the underlying CJDNS routing mechanism towards the creation of the overall* ***RAINBOW trust overlay mesh network*** *(Chapter 6).*

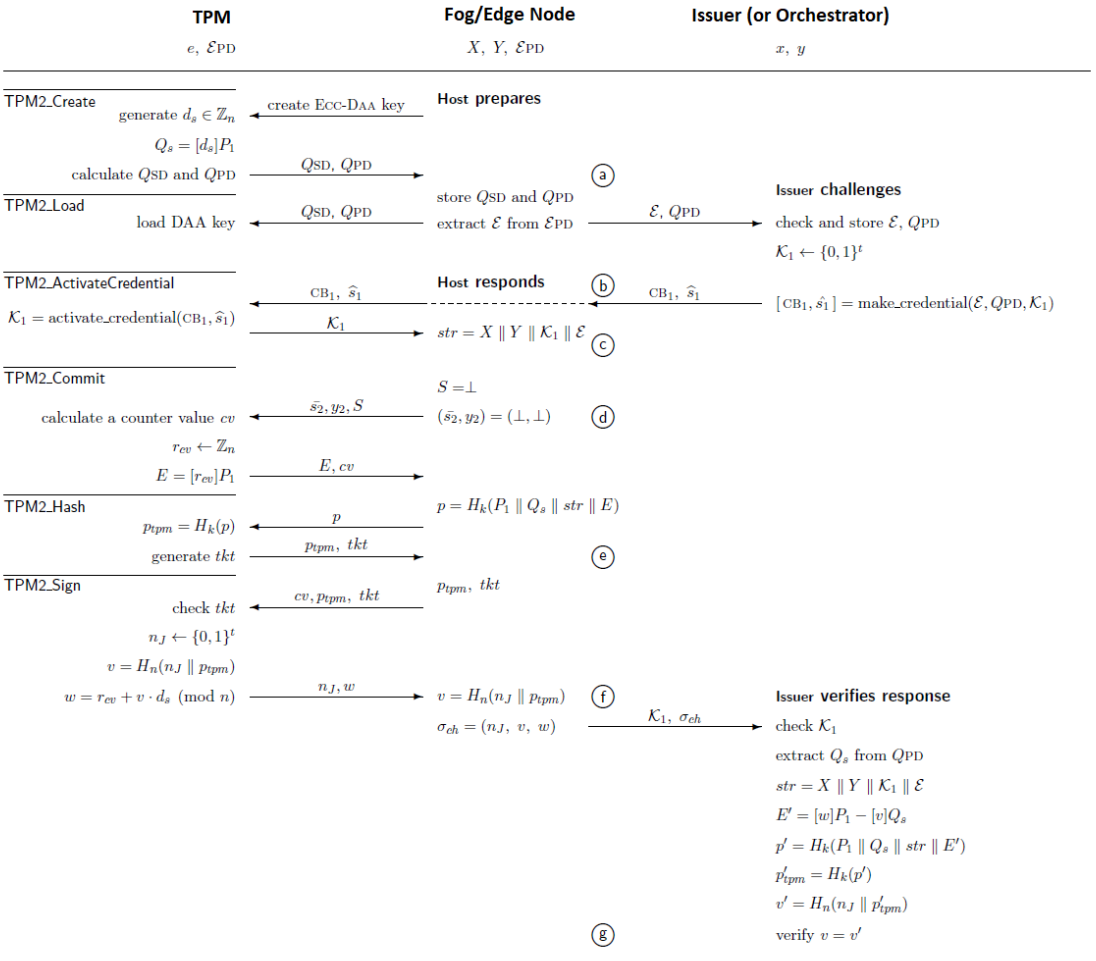

Figure 4.5: DAA Initiating the JOIN Phase (SETUP)

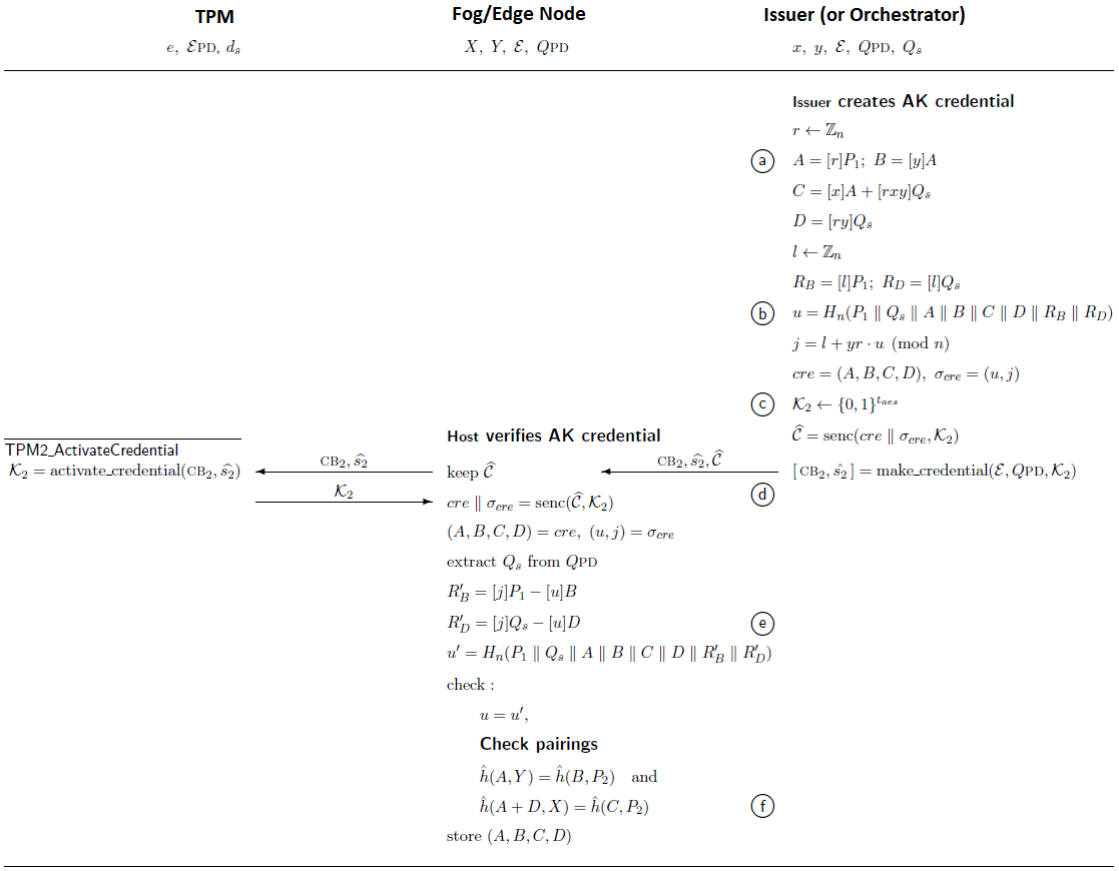

Figure 4.6: DAA Completing the JOIN Phase

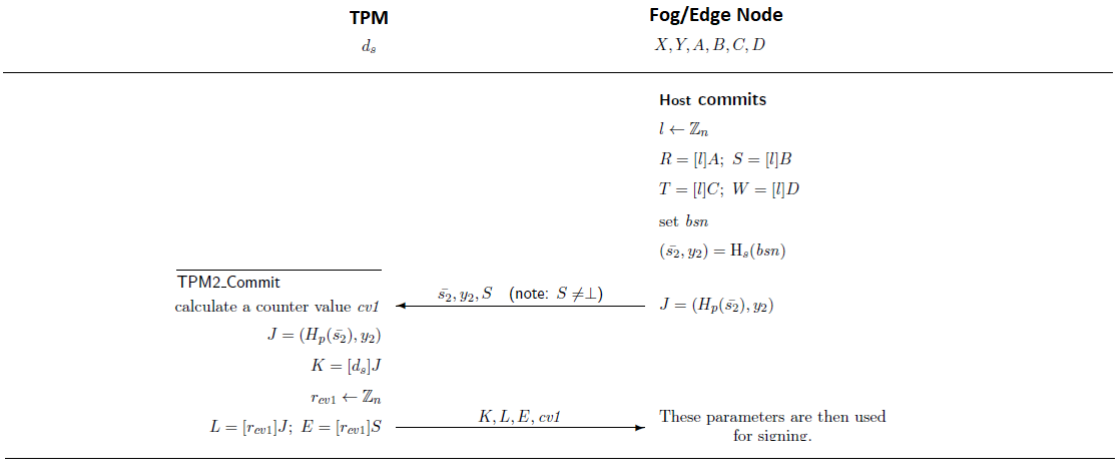

Figure 4.7: DAA Key Creation

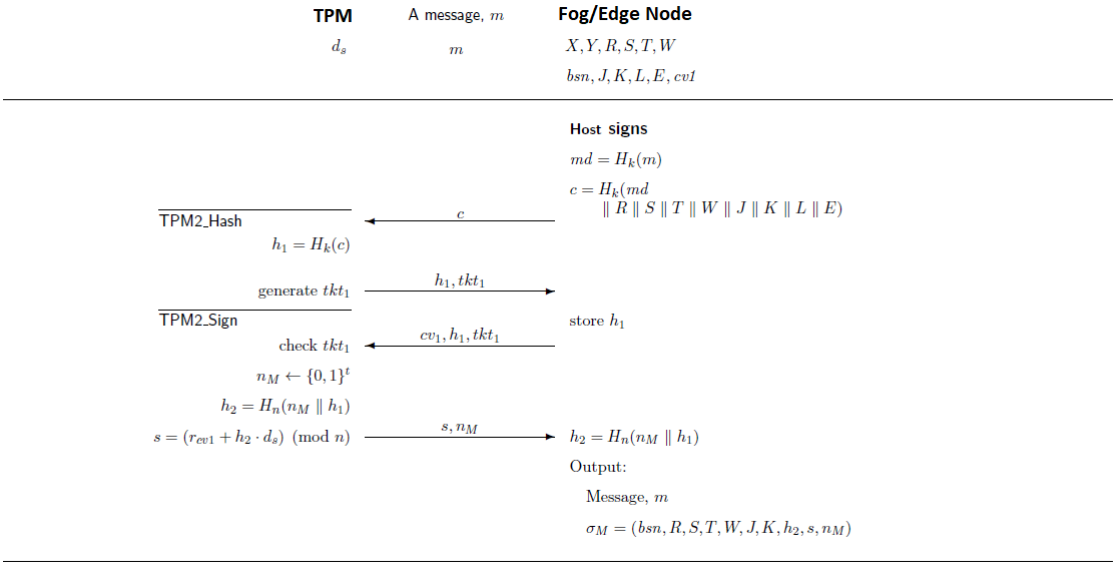

Figure 4.8: DAA SIGN Phase

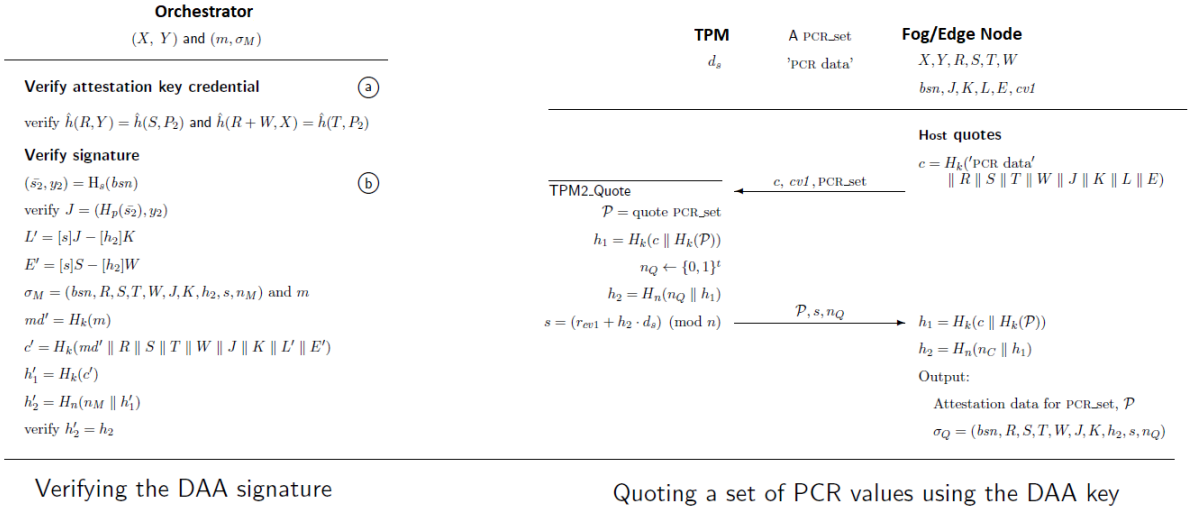

Figure 4.9: DAA VERIFY Phase & DAA Node Quote

Chapter 5

Anonymous Secure Channel Establishment

As described before, Direct Anonymous Attestation is an anonymous digital signature mechanism that allows for **secure platform authentication** and **privacy-preserving communication**. However, one of the main challenges in the integration of this DAA protocol, as part of the overall RAINBOW mesh networking stack (Chapter 6), **is the assumption of the existence of a secure communication channel during the $SETUP$ and $JOIN$ phases**. This basically puts a significant constraint to the fog/edge node enrollment process since it needs to execute over a secure channel for the DAA and Attestation Keys certification as well as the secure exchange of the *attestation policies* needed by the configuration integrity verification scheme (Chapter 3).

Compounding this issue, in RAINBOW we are planning to enhance this DAA protocol and try to solve the assumption that the Signer and the Issuer have established an one-way authentic channel [17], i.e., *anonymous and secure exchange of the parameters for the pairings that are executed for the creation of the ECC key, and the activate credentials step*.

5.1 On Using TLS keys: Benefits are Real, but so Are Drawbacks

One obvious solution would be the establisment and usage of TLS communication channels. Transport Layer Security, previously called Secure Socket Layer, is a cryptographic protocol tailored to ensure security, data integrity, trustworthiness, malware prevention and granular control of transmitted data over the the Internet. In short, it provides authentication, confidentiality, and integrity. The newest version is TLS 1.3 [41]. TLS 1.3 starts with a key exchange phase where the client and the server choose a random number r (256 bits fresh nonce), a list of favored symmetric cipher-suites, and an ephemeral (EC)DH private key. The private key belongs to one TLS DH group. The client is allowed to send several keys and the server takes up the responsibility to pick the group to continue. **After this phase the keying material, cryptographic parameter, and a encrypted channel is established.**

The next phase is the **authentication phase**. The server sends its certificate (X.509), with appropriate credentials/tokens that can be used for its efficient verification (signature over the transcript of the entire handshake messages using the private key), encrypts the extensions and terminates (HMAC over the transcript and the derived key). The client replies with “Secure ACK” to the server. All messages in the subsequent communication are now protected by AEAD with the derived key; this constitutes the third phase [41].

As with almost every protocol, TLS has some drawbacks when it comes to its applicability to various application domains other than HTTP and even with HTTP. Most noteworthy are high **latency, Men in the Middle (MiM) Attacks, network complexity, platform support, and implementation costs when using a certificate that is not free available**. Due to the widespread of TLS, it is a common attack surface to break the integrity and security. Here, ordinary attacks are Ciphersuite rollback attack [48], change chipher spec dropping attack in [48], version rollback attack [48], RSA-based sessions attack [36], adaptive chosen ciphertext attack on PKCS#1 [15], timing attacks [39], Cross-protocol attack based on ECC key exchange [37], SKIP-TLS attack [14], Factoring Attack on RSA-EXPORT Keys attack [1], Logjam attack [7], etc.

5.2 Key Exchange with Anonymous Authentication

In RAINBOW, our protocol aims to be more lightweight than TLS as well as provide different levels of *device- or user-controlled anonymity*. **We are using an ephemeral/static ECDH key agreement protocol with anonymous authentication between a Prover (P) and Verifier (V)**. This is based on the notation from Chen et al. [20] which also leverages the cryptographic primitives used in the DAA protocol.

The Prover has a Camenisch-Lysyanskaya credential [18] and a private key. She randomizes the credential and sends it to the Verifier who in turn verifies the randomized points. The Verifier then calculates a random point and the shared secret which is then sent back to the Prover. This parameter is subsequently used for calculating the same shared secret as the Verifier.

Both parties have now agreed on a common shared secret which can be used for deriving symmetric session keys KMAC, i.e, after receiving a message with KMAC integrity from the Prover (P), the Verifier (F) can assume that P has the private key belonging to the ticket.

An early draft of this described protocol is shown in figure 5.1[1]. On a successful run of this key agreement process, both parties know a common shared secret but they are not aware of any other identifying information of the other entity; thus, achieving **anonymous node authentication** which is a core requirement in fog-based environments and especially in safety-critical applications as the ones (for example) envisaged in the defined use cases (e.g., Urban Mobility). **This property will allow the (on-demand) privacy-preserving enrollment of the devices in a fog cluster**, thus, leading to an enhanced DAA $JOIN$ phases where a node (together with the underlying TPM) can verify the validity of its root of trust and unique device identity, to the $\mathcal{O}$rc, without revealing the actual identity of the attached TPM (i.e., certification of endorsement key in a privacy-preserving manner). Furthermore, a third party (e.g. an attacker) can neither learn the shared secret, nor information on the identity of the parties.

The proposed scheme is expected to offer a number of benefits, such as **no dedicated HW-TPM is required**, can be implemented in IoT Devices that offer interfaces that are not compatible with trusted components (e.g., Java card), **improved performance** and **privacy** in relation to TLS.

5.3 Research plan for Establishing Ephemeral Keys with Diffie-Hellman Key Agreement Protocol

The security mechanisms proposed for the RAINBOW platform are of different degrees of maturity. On one hand, TPMs are well-established with a large research body and a number of com-

[1] The finalization of the concrete models and implementation aspects will be provided in the context of D2.4

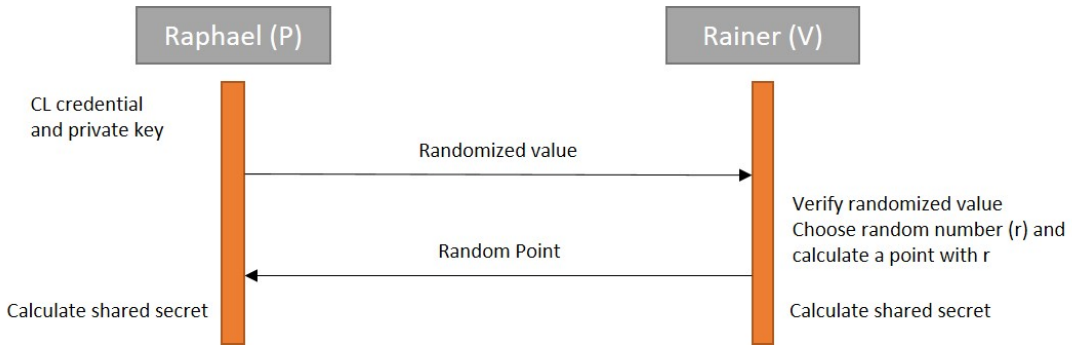

Figure 5.1: ADHKE - Anonymous Diffie-Hellman Key Exchange

mercial vendors offering such devices and solutions. On the other hand, the proposed Ephemeral Diffie-Hellman Key Agreement Protocol is a research proposal by IFAT which is at an early stage, but well-worth being examined in the course of RAINBOW.

For the automated, anonymous establishment of DH-Keys in RAINBOW, the following design and implementation plan will be followed:

1. Establishment of the mathematical scheme in detail, e.g., definition of protocol roles and actions, pre-conditions and security claims and assumptions. Similar to the models that have already been compiled for the RAINBOW attestation enablers presented in the previous chapters.
2. Evaluation of the protocol axioms, security analysis and discussion in the RAINBOW threat model.
3. Implementation of the scheme within the RAINBOW-configured Raspberry Pi platforms as a proof-of-concept. The first version will be based on the DAA cryptographic primitives.
4. Classification of the resource and performance measurements coupled with a detailed evaluation.
5. Study of the applicability of hardware-based cryptography accelerators.
6. Integration in the RAINBOW demonstrator.
7. Further research on the extension of the basic scheme towards a a direct pseudonymous key agreement protocol and a direct pseudonymous encryption/decryption scheme.

Considering the early stage of the protocol, the above mechanism will be instantiated in the second version of the RAINBOW security and trust enablers. The above tasks will documented in D2.5.

As contingency plan, the RAINBOW middleware may fall back to conventional TLS based key agreement protocols, which might incur a performance overhead over the proposed protocol.

Chapter 6

Integration of CJDNS Protocol in the RAINBOW Stack

In what follows, we present the underpinnings of the **CJDNS routing mechanism** that is leveraged (and enhanced) by RAINBOW to act as an umbrella layer that will facilitate the secure node interaction based on all the aforementioned secure enrollment, attestation and system assurance protocols. *We have to note that in this deliverable the focus is on presenting the **logistics, mode of operation, and workflow of actions** of the CJDNS protocol and not the complete RAINBOW version with all the TPM-based security components integrated; this will be described in D2.4.*

6.1 The Necessity of Mesh Networking in RAINBOW

The deployment of wireless fog nodes in an uncontrolled manner (without supervision entities) raises many functional requirements as far as connectivity is concerned. On the one hand, wireless nodes formulate **temporal connections** with its **adjacents**. In addition, nodes must route packets to each other **without relying on static routing tables** and **fixed network subnets**. In other words, the networking environment per se is dynamic and uncontrolled. Each node that enters a wireless fog network must be **addressable**. It goes without saying that in fixed networks the addressing-problem is solved either with static configuration or through auto-configuration that is mandated by a DHCP server.

In mesh networks, none of these addressing schemes are applicable. The lack of **static subnetting makes it impossible to bind a specific IPv4/IPv6 to a node** since a node may participate (based on its mobility pattern) in **many subnets**. In addition, even if a node decides by itself an IP address, it has no guarantees that its address will not **suffer from collision** during an opportunistic encounter with another node.

Beyond the problem of addressing, **the problem of routing** is even more intense. More specifically, the global internet expansion relies on **well established protocols** (e.g. BGP) that can route a packet from one node to another using hierarchical routing schemes that logically interconnect various Autonomous Systems (ASs). Hence, the **"primitive" functions** of routing such as *path establishment*, *acknowledge of delivery* etc. are considered granted. However, in mesh networks, paths cannot be statically computed and stored beforehand due to the dynamicity of the environment.

Finally, the problem of **trustworthiness** of a node that joins a mesh environment is prominent. A node that joins an "overlay" may act as a curious interceptor of packets that are routed through this node. Hence, it goes without saying that all aspects of a node (i.e. running applications,

operating system, firmware) should provide integrity evidence in order to be immune on several cyber attacks. In non-fog deployment, several of the attack vectors that we want to eliminate do exist, however someone could claim that a node is joining a network in a supervised manner and as such several pre deployment integrity checks can be performed (although, theoretically, these checks do not provide strong runtime integrity guarantees).

In the frame of RAINBOW, **unambiguous addressing, reachability and trust** are in the epicenter of the **zero-touch-configuration requirements**. In other words, the framework should provide guarantees that a node will be able to automatically participate in a dynamic routing scheme in order to expose/advertise its computational resources and participate in analytics processes. The RAINBOW centralized controller should be able to "pull" instrumentation-measurements and as such the connectivity of the controllers with the nodes should be considered granted. To do so, a forked version of an existing mesh-routing stack will be delivered. The stack that will be extended is CJDNS which will be analyzed below.

6.2 Fundamentals of CJDNS

CJDNS is a mesh protocol that solves out of the box issues such as **global addressing and connectivity**. This means that any CJDNS-enabled node can interconnect with any other CJDNS node automatically, without central authority or control. Moreover, all CJDNS-enabled networks (hereinafter addressed as **meshnets**) are compatible by their very nature. In fact, there is really only ever one single global CJDNS meshnet, even if some parts of it are not linked to some others. **The moment they are linked, they will function as one**. Furthermore, CJDNS also includes secure end-to-end encryption built in to the protocol at the very lowest levels. In fact, the encryption is part of what allows for the global distributed addressing. When a **new CJDNS node is set up, a cryptographic key pair is generated and the node's IP address is derived from that key**. Any communication to the node is automatically encrypted with that key, and communications **with any other IP address can be cryptographically verified as secure and genuine by comparing the keys used to the address itself**. What this all means is that nobody on the meshnet can see your private communications except for you and the node you are actually communicating with.

The distribution and management of such cryptographic keys and certificates is achieved through the use of appropriate Public Key Infrastructures (PKIs). While intensive research efforts have proven the security guarantees provided in such architectures, there are a number of challenges inherent to PKIs (and, thus, CJDNS) when it comes to **privacy** (*no privacy requirements are currently been considered by the CJDNC mesh networking stack*), **scalability**, and **operational assurance** [29]. In its core, the possibility of security breaches has the potential to seriously weaken the technical security protection measures of CJDNS, since in its current version the assumption is on the existence of a number of centralized trusted entities. However collusion or security incidents affecting certification authorities have grown more frequent in the recent past [26], so the existence of a PKI architecture does not guarantee per se the enactment of trust between the peers and additional measures are necessary to reinforce a scalable community of trust [9].

Another interesting feature of CJDNS is it's **efficient routing**. Because it's designed to have **lower resource requirements (primarily memory)** than traditional internet routing, CJDNS uses a system of routing that **minimizes the amount of information a router needs** to do its job. A side benefit of this is that no individual node knows any more about who you are communicating

with then it absolutely needs to, which generally means it only knows **what the next hop is along the path, not the final destination**. This further enhances privacy, beyond what is even possible on the internet without additional specialized tools like Tor. It should be clarified, though, that CJDNS does not offer actual anonymity anywhere near the level that Tor does, nor is it intended too. It does, however, offer just enough to make mass surveillance impractical, while not sacrificing performance like Tor does.

It should be noted that CJDNS is a **pure "Layer 3" protocol** that runs directly **on top of the MAC layer**, intended as a replacement for the **standard TCP/IP protocol** used in today's network and internet connectivity. It is **compatible** with **plain direct ethernet or ad-hoc wireless connection**. It actually implements TCP/IP on top of itself and offers a **standard IPv6 interface to applications**. In other words, **existing layer-7 applications** (as the ones deployed by RAINBOW) can work **without modification**, provided they support IPv6 since it does not rely on any other meshnet or internetworking protocols to function.

6.3 Layering of CJDNS

CJDNS is made of **three major components** which are tightly integrated together (see Figure 6.1). There is a **switch, a router, and a CryptoAuth module**. With total disregard for the OSI layers, each module is inherently dependent on both of the others. The router cannot function without routing in a small world which is made possible by the switch, the switch is blind and dumb without the router to command it, and without the router and switch, the CryptoAuth has nothing to protect.

6.3.1 The Switch Component

The switch design is unlike an IP or Ethernet router, it doesn't need to have knowledge of the globally unique endpoints in the network. Like **ATM switching, the packet header is altered at every hop** and the **reverse path can be derived at the end point** or at any point along the path but **unlike ATM, the switch does not need to store active connections and there is no connection setup**.

In order to optimise the protocol description let us provide some definitions that we be used during the overall description:

- **Interface:** A point-to-point link to another cjdns switch. This may be emulated by Ethernet frames, UDP packets or other means.
- **Self Interface:** A special Interface in each switch. Packets sent for this interface are intended for the node which this switch is a part of. Upon reaching the ultimate hop in its path, a packet is sent through the Self Interface so it can be handled by the next layer in the node.
- **Director:** A binary codeword of arbitrary size which when received by the switch will direct it to send the packet down a given Interface.
- **Route Label:** An ordered set of Directors that describe a path through the network.
- **Encoding Scheme:** The method by which a switch converts one of its internal Interface ID (EG: array index) to a Director and converts a Director back to its internal representation.

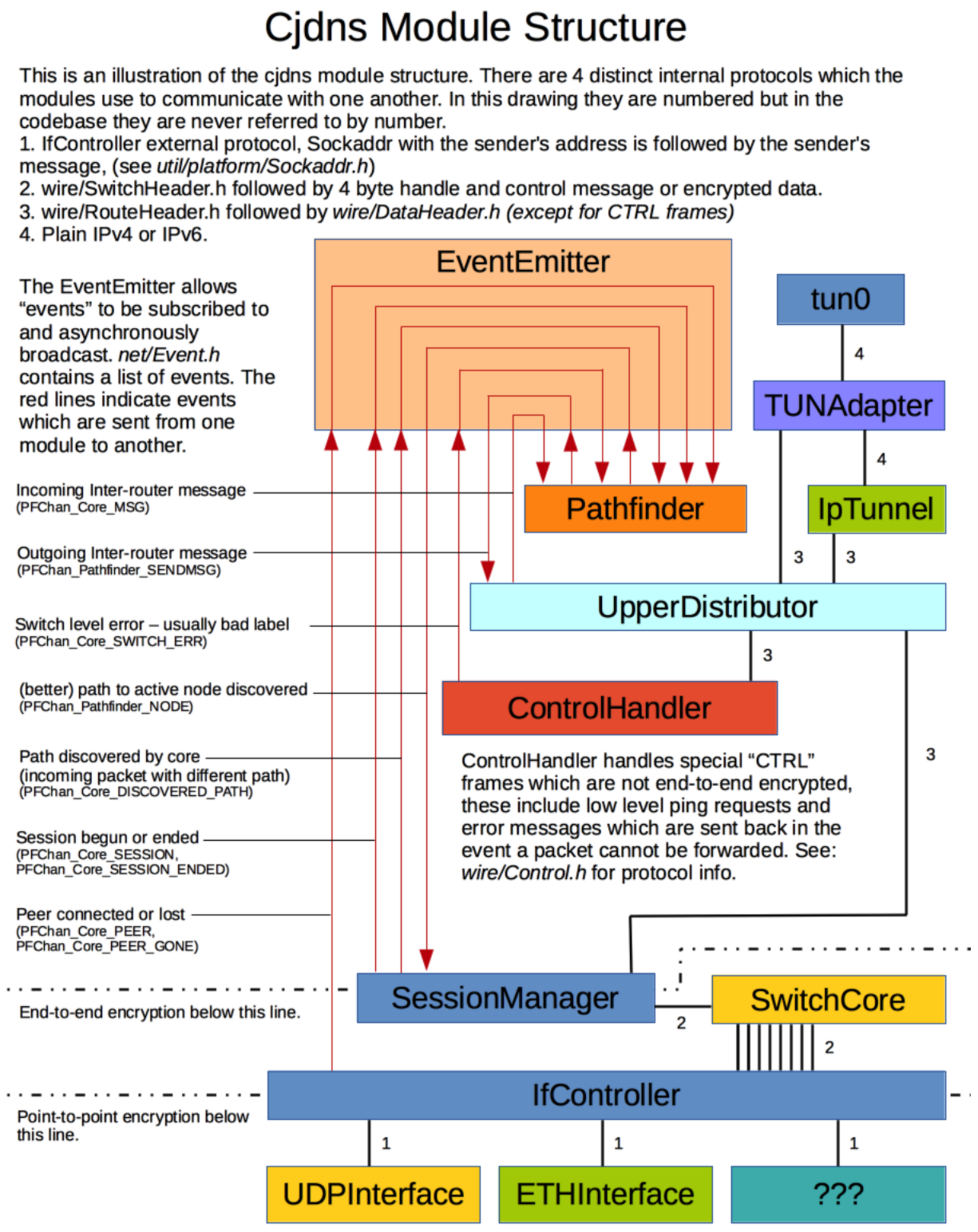

Figure 6.1: Layering of the CJDNS Stack

Encoding schemes may be either fixed width or variable width but in the case of variable width, the width must be self-describing as the Directors are concatenated in the Route Label without any kind of boundary markers.

- **Encoding Form:** A single representation form for encoding of a Director. For a given Encoding Form, there is only one possible way to represent a Director for a given Interface. A variable width Encoding Scheme will have multiple Encoding Forms while a fixed width Encoding Scheme will have only one.
- **Director Prefix:** For switches which implement variable width encoding, the least significant bits of the Director is called the Director Prefix and is used to determine the width of the Director. It should be clarified, that because the Route Label is read from least significant bit to most significant bit, the Director Prefix is actually the bits furthest to the right of the Director.

Now let us examine an **indicative packet switching flow**. When a packet comes into the switch, **the switch uses its Encoding Scheme to read the least significant bits** of the Route Label in order to determine the Director and thus the Interface to send the packet down. The Route Label is shifted to the right by the number of bits in the Director, effectively ***removing*** **the Director and exposing the Director belonging to the next switch in the path**. Before sending the packet, the switch uses its Encoding Scheme to craft a Director representing the Interface which the packet came from, does a bitwise reversal of this Director and places it in the empty space at the left of the Route Label which was exposed by the previous bit shift. In this way, the switches build a mirror image of the return Label allowing the endpoint, or any hop along the path, to derive the return path by simple bitwise reversal without any knowledge of the Encoding Schemes used by other nodes.

6.3.2 The Router Component

A router has **3 functions**: **a) it periodically searches for new nodes**, **b) it responds to node searches**, and **c) it forwards packets**. When a router responds to a search, it responds with nodes which it thinks will get closer to the destination. The **responses must not have addresses which are, in address space distance, further from the responding node than the search target**, and they must not have routes which begin with the same interface as the route to the querying node. These two simple rules provide that **no search will ever go in circles and no route will ever go down an interface, only to be bounced back**. While the second rule can only be enforced by the honor system, querying nodes MUST double check the first rule. The node doing the searching adds the newly discovered nodes to their routing table and to the search, then continues the search by asking them.

Upon receiving a **search response** containing one's own address, **a node should purge** all entries from its table whose routes begin with that route. This will control the **proliferation of redundant routes**. The **"address space distance"** between any two given addresses is defined as the result of the two addresses XOR'd against one another, rotated 64 bits, then interpreted as a big endian integer. The so called "XOR metric" was pioneered in the work on **Kademlia DHT** [38] system and is used to forward a packet to someone who probably knows the whole route to the destination. The 64 bit rotation of the result is used to improve performance where the first bits of the address is fixed to avoid collisions in the IPv6 space.

Adding nodes to the routing table from search responses is done by splicing the route to the node which was asked with the route to the node in the response, yielding a route to the final destination through them. **Routers choose the node to forward a packet to in a similar way to how they answer search queries**. They select nodes from their routing table except in this case the selection contains only one node. The packet is sent through the CryptoAuth session

corresponding to this node and the label for getting to it is applied to the packet before sending to the switch. The ”search target” for forwarding a packet is the IPv6 destination address of the packet.

6.3.3 The CryptoAuth Component

The CryptoAuth is a mechanism for **wrapping interfaces**, i.e., it is provided with an **interface and optionally a key**, and it provides a **new interface which allows to send packets to someone who has that key**. Like the rest of CJDNS, it is designed to function with best effort data transit. **The CryptoAuth handshake is based on piggybacking headers** on top of regular data packets and while the traffic in handshake packets is encrypted and authenticated, **it is not secure against replay attacks and has no forward secrecy if the private key is compromised**. The CryptoAuth header adds takes 120 bytes of overhead to the packet.
There are **5 types of CryptoAuth headers**:

1. **Connect-To-Me** header which is used to start a session without knowing the other node’s key. If ”Session State” is equal to the bitwise complement of zero, the sender is requesting that the recipient begins a connection with him, this is done in cases when the initiator of the connection does not know the key for the recipient. If the entire header is not present the recipient must drop the packet silently, the only field which is read in the packet is the ”Permanent Public Key” field, all others SHOULD be ignored, specifically, content must not be passed on because it cannot be authenticated. The recipient of such a packet should send back a ”hello” packet if there is no established connection. If there is already a connection over the interface, the recipient should not respond but may allow the connection to time out faster.

2. **Hello Packet** header which is the first message in the beginning of a session. If the ”Session State” field is equal to zero or one, the packet is a Hello Packet or a repeated Hello Packet. If no connection is present, one may be established and the recipient may send a Key Packet in response but it is recommended that he wait until he has data to send first. A node who has sent a Hello Packet, has gotten no response and now wishes to send more data must send that data as more (repeat) Hello Packets. The temporary public key and the content are encrypted and authenticated using the permanent public keys of the two nodes and ”Random Nonce” in the header. The content and temporary key is encrypted and authenticated using **curve25519-poly1305-salsa20** function, using the shared secret computed as described in the Authentication field’s section.

3. **Key Packet** header which is the second message in a session. If the ”Session State” field is equal to two or three, the packet is a Key Packet. Key Packets are responses to Hello Packets and like Hello Packets, they contain a temporary public key encrypted and authenticated along with the data. Once a node receives a Key Packet it may begin sending data packets. A node who has received a Hello Packet, sent a Key Packet, gotten no further response, and now wishes to send more data MUST send that data as more (repeat) key packets.

4. **Data Packet** header- A traffic packet with **Poly1305**[1] authentication: The Data Packet is the default data packet. **The first 4 bytes are used as the nonce**, in this case it is a 24 byte nonce and curve25519-poly1305-salsa20 is used to encrypt and decrypt the data, using

[1] https://cr.yp.to/mac.html

the shared secret computed using one peer's temporary public key and the other peer's temporary secret key (both roles are symmetrical and produce the same shared secret).

5. **Replay Protector** header that safeguards a session: The replay protector is a feature cjdns implementations provide. It is however not part of the protocol itself.

As described in Chapter 7, the end goal is to enhance this workflow of actions and CryptoAuth component, with the integration of the DAA mechanism (Chapter 4) so as to enable enhanced operational assurance and privacy-preserving communication between fog nodes. This is considered as one of the main goals towards **"security and privacy by design"** solutions, including all methods, techniques, and tools that aim at enforcing security and privacy at software and system level from the conception and guaranteeing the validity of these properties.

6.4 Cross-Component Packet Processing

The journey of a packet begins at the user interface device (TUN or similar). The user sends an IPv6 packet which comes in to the TUN device and enters the engine, it is checked to make sure its source and destination addresses are valid and then a router lookup is made on the destination address. cjdns addresses are the first 16 bytes of the SHA-512 of the SHA-512 of the public key. **All addresses must begin with the byte 0xFC** otherwise they are invalid, generating a key is done by brute force key generation until the result of the double SHA-512 begins with 0xFC.

After the router lookup, **the node compares the destination address to the address of the next router**, if they are the same, the inner layer of encryption is skipped. Assuming they are different, the IPv6 header is copied to a safe place and a CryptoAuth session is selected for the destination address, or created if there is none, and the packet content is passed through it. **The IPv6 header is re-applied on top of the CryptoAuth header for the content**, the packet length field in the IPv6 header is notably not altered to reflect the headers which are now under it.

The packet is now **ready to send to the selected router**. For sending the packet to the router, **a CryptoAuth session is selected for the router's address and the packet, from IPv6 header down**, is **passed through it**. A switch header is applied to the resulting encrypted structure and it is sent down to the switch for routing.

The switch takes the packet and sends it to a network module which **uses yet another CryptoAuth session to encipher and authenticate the packet from the switch header down**. The resulting data is packaged in a network packet and sent to the switch at the next node. Upon receiving the packet, the next node **sends the packet through its CryptoAuth session thus revealing the switch header and it sends the packet to its switch**. The switch most likely will send the packet out to another endpoint as per the dictate of the packet label but may send it to its router, eventually the node for which the packet is destined will receive it.

The router, upon receiving the packet will examine it to see if it appears to be a CryptoAuth Connect To Me packet, Hello packet, or Key packet. If it is one of these, it will insert the IPv6 address, as derived from the public key in the header, into a hashtable so it can be looked up by the switch label. Otherwise it will do a lookup. If the Address cannot be found in its hashtable, it will try asking the router if it knows of a node by that label and if all fails, the packet will be dropped.

From the IPv6 address, it will lookup the CryptoAuth session or create one if necessary, then pass the opaque data through the CryptoAuth session to get the decrypted IPv6 header. If the

source address for the packet is the same as the double SHA-512 of the public key for the router from which it came, it's assumed to have no inner layer of encryption and it is written to the TUN device as it is. If its source address is different, it is passed back through a CryptoAuth session as selected based on the source IPv6 address. The IPv6 header is then moved up to meet the content (into the place where the CryptoAuth header had been) and the final packet is written out to the TUN device.

```
"nodeX:19071":
                {
                "password": "public_JFh4rX0R1jm6a7eKWCzD",
                 "publicKey": "425bcpr9ns0jpuh9ffx1lbbktkd3tpln16jzs9sgbjdkvfg25zv0.k"
                }
```

Figure 6.2: Static Configuration of Trusted Peer

6.5 Admission Control

Up to now it should be absolutely clear that CJDNS, can support, functionally, any high-level application (layer 7) that is compatible with traditional IPv6 sockets. However, the admission to the network can be separated into two distinct modalities which are **Layer-3 admission and Layer-2 admission**. We will shed light to both of these modalities in order to provide insight of the engineering that has to be performed in the frame of RAINBOW.
Layer-3 admission implies that one node that aims to join an overlay network has to **explicitly declare a trusted node**, which means that an **"offline" exchange containing a public-key and a peer password must be performed**. It is prominent, that both parties (i.e. peers) have to update their configuration with a node-config as depicted in Figure 6.2.

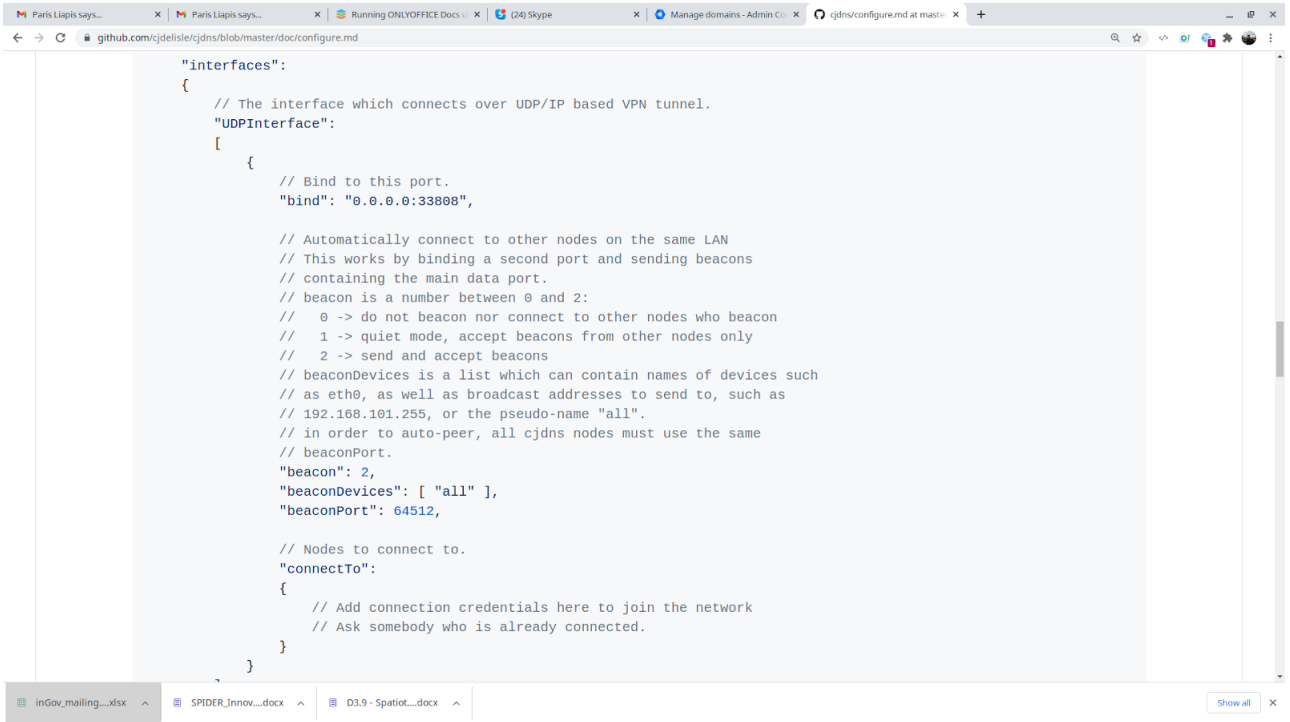

Figure 6.3: Blink Acceptance Configuration

It could be argued that this option suffers from **severe scalability issues**. I.e. it cannot scale in large deployments since it requires static/manual configuration at the peer level. However, in the frame of RAINBOW layer 3 admission is not important since it is applied only to stationary nodes (Data Center resources, centralized cluster-heads etc.).

Layer-2 admission implies that two nodes that are in **physical proximity to each other can self-pair**. The way this is performed in the current version is through a "blind acceptance mechanism" (an indicative blind acceptance configuration is presented on Figure 6.3). According to this mechanism some nodes of an existing network act as **pairing-acceptor**, i.e., they are configured to accept Layer-2 beacons from nodes that are not part of the network; but they wish to join.

On the other hand, nodes that wish to join are configured to emit/solicitate in a **flooding manner** pairing requests in order for pairing acceptors to receive them. Upon reception of pairing request an acceptor is initiating a key-exchange protocol. Ultimately, the two nodes exchange their public keys and create mutual authentication credentials that is **blindly appended to their configuration**. This configuration is used to trigger CryptoAuth sessions which have been extensively analyzed above. **The blind acceptance protocol of CJDNS will be heavily engineered in order a) to rely on strong trust anchors (e.g. TPM) and b) to be complemented by an attestation process in order to leverage the security guarantees.**

Chapter 7

Towards Secure & Privacy-Preserving Overlay Mesh Networking

Based on the aforementioned (decentralized) security and privacy anchors, the last crucial step for fulfilling the **main vision of RAINBOW on establishing a trusted fog computing architecture**, is the integration of all these trusted computing extensions towards the establishment and management of **trust-aware service graph chains**. In this context, *the communication over the continuum from edge devices to fog nodes and backend cloud systems must support secure interactions between all participating entities in order to establish **service-specific "fog/edge node communities of trust".***

To do so, as has been described previously, RAINBOW is leveraging advanced cryptographic primitives (Direct Anonymous Attestation (Chapter 4) for privacy) and enhanced remote attestation (Chapter 3) for security and operational assurance. At a conceptual level, the goal is to enable fog/edge entities to establish and maintain trust during the entire system life-cycle. *This stems from establishing **roots of trust** in components (by leveraging the attached TPM), and using these roots of trust to establish and maintain trust relationships.*

However, in the road towards the establishment of such trust-aware SGCs, **fog/edge node interaction is a challenge by itself** since the target environment is **dynamic** and **uncontrolled** (Section 6.1). More specifically, a RAINBOW deployment consists of nodes that formulate temporal connections. In addition, nodes must route packets to each other without relying on static routing tables and fixed network subnets. On top of that, there are two additional crucial constraints that need to be taken into consideration. The first is the **lack of a network addressing scheme** and the second is the **establishment of trust** among the nodes that participate in the fog deployment towards the creation of "communities of trusted devices" that can enable the secure community communications and can then be used for the trusted deployment of the envisioned services.

Addressing these challenges lies in the heart of RAINBOW towards the provision of a **secure overlay mesh network for delivering the high-level functionalities related to secure (edge and mesh) device identification and integrity, data integrity and confidentiality, anonymity and resource integrity** as described in the overall framework reference architecture (see Deliverable D1.2 [21]). In what follows, we provide an overview of the conceptual architecture and workflow of actions - of the entire RAINBOW trust overlay mesh network - that will merge all the previously described trust extensions, in an enhanced CJDNS networking stack, and that will be described in the context of Deliverables D2.3 and D2.4.

7.1 Design Choices & Benefits

The integration of trusted computing technologies into the fog-based ecosystem allows for the establishment of much stronger end-to-end chains of trust that can be used according to the needs of all involved parties. The primary benefits of such a decentralized solution are in terms of **security, privacy and scalability**.

Figure 7.1 depicts an abstract description of all internal building blocks and interfaces of the RAINBOW endgoal when it comes to secure overlay mesh networking. As it can be seen, the initial step **(Step 0)** is to enable the **secure enrollment of the hardware and perform a Zero-Touch Configuration to each of the fog participants and the belonging IoT devices**. After the established enrollment and setup phase, the Orchestrator ($\mathcal{O}$rc) initiates the privacy configuration for each participant through the establishment of respective **DAA primitives (Step 1)**. Then, the fog nodes and IoT devices are capable of establishing an **anonymous secure channel (Step 2)** and authenticate themselves with **privacy protection (Step 33)**. **If these steps are successfully performed, the devices are able to join the mesh network (Step 4)** and they are allowed to interact with each other in the system with the registered services from the RAINBOW platform and can enforce a **secure CJDNS routing (Step 5)**.

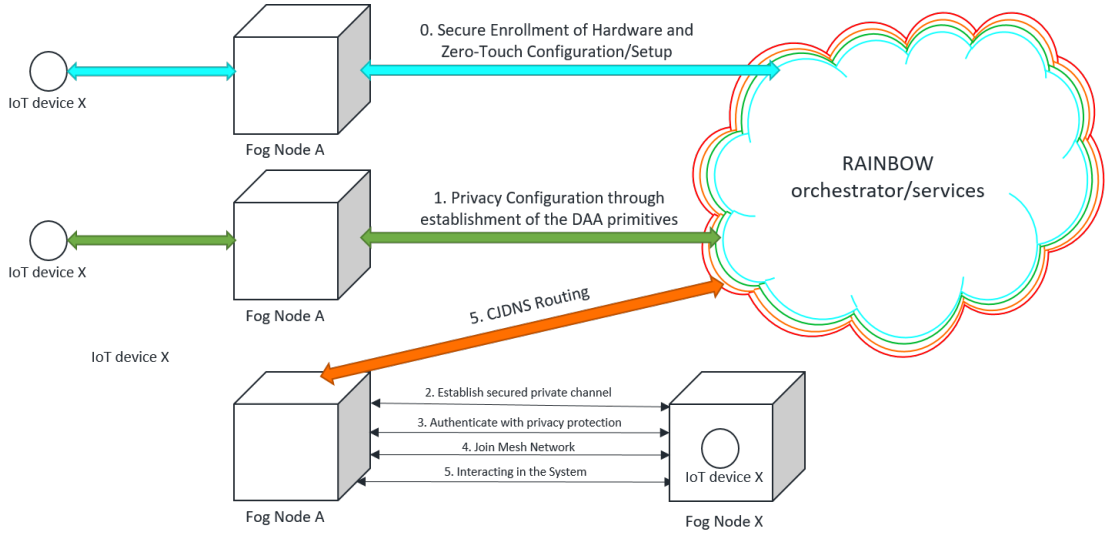

Figure 7.1: System components interaction within RAINBOW

One of the biggest advantages of such a decentralized approach is its scalability, as **trust is shifted from the back-end infrastructure to the fog nodes and IoT edge devices**. Applying the integrity verification and DAA protocols results in the redundancy (and removal) of the most of the infrastructure entities currently assumed in traidtional PKI-based architectures: edge devices can now create their own pseudonyms, and DAA signatures are used to self-certify each such credential that is verifiable by all verifiers. Furthermore, nodes have total control over their privacy, as no trusted third-party is involved in the creation phase of the short-term anonymous credentials. This means that it is infeasible for any third-party to reveal the identity of another node assuring that identity resolution is not possible in such a solution.

Analyzing the requirements specified in Section 4.1, it is clear that all necessary properties are achieved with the addition of security and user-controlled privacy. The *anonymity*, *pseudonymity* and *unobservability* properties are built into DAA's algorithms, JOIN and SIGN / VERIFY by using anonymous digital signatures. Therefore, third-parties cannot identify and link subsequent service

requests originating from the same fog node. This is also true in the presence of colluding third-parties and other infrastructure entities. The JOIN protocol is intentionally not privacy-preserving as the $\mathcal{O}$rc needs to be aware of the node to be authenticated. However, successful completion of the protocol results in the node solely owning a DAA credential.

Unlinkability (and/or different levels of *vehicle linkability*) is controlled by the node through the DAA SIGN / VERIFY phases. A node has control over its DAA credential, and can decide whether or not to "blind" it, thus, producing pseudonyms (and revocation) that are linkable. The proposed approach provides privacy-preserving linkability via DAA deterministic signatures, where the use of a pseudonym is unlinkable to any other pseudonyms owned by a node. This property is of particular interest to fog-based services as nodes can demonstrate unobservability and unlinkability (when using multiple services) while being accountable for these service invocations.

In addition, DAA also provides *non-frameability* and *correctness* properties which are security attributes that state-of-the-art solutions do not capture entirely. DAA ensures that only valid and trustworthy TPMs are able to join the network by ensuring that the endorsed TPM keys have not been previously compromised. This ensures that TPMs only produce valid signatures and can only be linked when specified by a particular authorized service.

Furthermore, the integration of such advanced integrity verification capabilities, enable RAINBOW to protect the overall system from adversaries trying to convince the Orchestrator that binaries have not been manipulated by exploiting either the quoting process of building a fraudulent certificate. The certificate comprises the current PCR values and the nonce from the $\mathcal{O}$rc. Assuming the accumulated PCRs reflect the adversary's presence, she can try to tamper with the certificate creation process to reflect a forged PCR digest. Unfortunately for the adversary, the RAINBOW trust extension will be reluctant to sign the forged certificate since it did not create it.

Based on the above, we denote our solution (to be modeled and implemented in the context of Deliverables D2.3 and D2.4) as **"zero-conf" as IoT service networking is a burden left to RAINBOW which pushes intelligence to the control plane for fog node discovery, coverage and self-healing** (after mesh network splitting/merging); all important properties for future 5G networks over heterogeneous and geo-distributed (fog) infrastructure.

Chapter 8

Conclusions

This final section will act as a synopsis of this deliverable and summarize its findings. The scope of this deliverable was to provide the **mode of operation, work-flow, and building blocks** of the newly introduced set of secure RAINBOW trust extensions; as part of the first release of the RAINBOW trust overlay mesh networking capabilities towards the estblishment of trust-awre service graph chains. This is considered as one of the main goals towards **"security and privacy by design"** solutions, including all methods, techniques, and tools that aim at enforcing security and privacy at software and system level from the conception and guaranteeing the validity of these properties. For privacy, RAINBOW leverages advanced crypto primitives, namely Direct Anonymous Attestation (DAA), whereas for security and operational assurance, it enables the provision of Platform Integrity Verification.

As part of the overall RAINBOW attestation toolkit, the main goal is to allow the creation of **privacy- and trust-aware service graph chains** (managed by the Orchestration Lifecycle Manager and estabslished by the RAINBOW Deployment Manager) through the provision of **zero-touch configuration functionalities**: *fog nodes, wishing to join a fog cluster, adhere to the compiled attestation policies by providing verifiable evidence on their configuration integrity and correctness.*

In this context, the RAINBOW collective attestation algorithms provide a multi-level security verification mechanism for supporting trust aware service graph chains (based on the identified security attestation policies) on the integrity assurance and correctness of the comprised devices: **from the trusted launch and configuration to the run-time attestation of low-level configuration properties.** Based on our analysis, we described how a device achieves privacy-preserving integrity correctness and how to utilize the attached TPM for binary data integrity. Our early implementation and evaluation results demonstrate that trust enablers can satisfy the privacy, security, and efficiency requirements.

Furthermore, by considering the salient characteristics of all internal RAINBOW secure components (i.e., Integrity Verification, DAA, Secure Anonymous Authenticated Communication Channel, and CJDNS Mesh Networking), we provided the conceptual architecture of the overall trust overlay mesh networking stack that will be provided in the upcoming deliverables. Overall, this RAINBOW trust overlay mesh network, will integrate all the appropriate trust extensions - embedding integrity verification and DAA crypto-primitives - for trust establishment enabled upon service deployment by the RAINBOW Orchestrator. This alleviates users from having to deal with secure network routing in fog environments and from having to establish trust among components and services and performing runtime verification and remote attestation.

References

[1] Factoring rsa export keys - freak (cve-2015-0204). https://access.redhat.com/blogs/766093/posts/1976563, 2015. Accessed: 2020-12-01.

[2] Trusted platform module (tpm) 2.0: A brief introduction. https://trustedcomputinggroup.org/wp-content/uploads/TPM-2.0-A-Brief-Introduction.pdf, 2015. Accessed: 2020-12-01.

[3] Ibm's tpm 2.0 tss. https://sourceforge.net/projects/ibmtpm20tss/, 2020. Accessed: 2020-12-01.

[4] Linux tpm2 tss2 software. https://github.com/tpm2-software, 2020. Accessed: 2020-12-01.

[5] Trusted platform module (tpm). https://trustedcomputinggroup.org/work-groups/trusted-platform-module, 2020. Accessed: 2020-12-01.

[6] Tigist Abera et al. C-FLAT: Control-Flow Attestation for Embedded Systems Software. In *Proceedings of the 2016 ACM SIGSAC CCS Conf.*, pages 743–754.

[7] David Adrian, Karthikeyan Bhargavan, Zakir Durumeric, Pierrick Gaudry, Matthew Green, J. Alex Halderman, Nadia Heninger, Drew Springall, Emmanuel Thomé, Luke Valenta, Benjamin VanderSloot, Eric Wustrow, Santiago Zanella-Béguelin, and Paul Zimmermann. Imperfect forward secrecy. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM, oct 2015.

[8] Will Arthur, David Challener, and Kenneth Goldman. *A Practical Guide to TPM 2.0*. Apress, 2015.

[9] Opinion 03/2017 on Processing personal data in the context of Cooperative Intelligent Transport Systems (C-ITS). Document, October 2017.

[10] Katelin A. Bailey and Sean W. Smith. Trusted virtual containers on demand. In *5th ACM Workshop on Scalable Trusted Computing*, STC '10, page 63–72, 2010.

[11] J. Christopher Bare. Attestation and trusted computing. https://courses.cs.washington.edu/courses/csep590/06wi/finalprojects/bare.pdf, 2006. Accessed: 2020-12-01.

[12] Elaine B. Barker and John M. Kelsey. Recommendation for random number generation using deterministic random bit generators. Technical report, jun 2015.

[13] Michael Till Beck and Juan Felipe Botero. Scalable and Coordinated Allocation of Service Function Chains. *Comput. Commun.*, 102, 2017.

[14] Benjamin Beurdouche, Karthikeyan Bhargavan, Antoine Delignat-Lavaud, Cedric Fournet, Markulf Kohlweiss, Alfredo Pironti, Pierre-Yves Strub, and Jean Karim Zinzindohoue. A messy state of the union: Taming the composite state machines of TLS. In *2015 IEEE Symposium on Security and Privacy*. IEEE, may 2015.

[15] Daniel Bleichenbacher. Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS #1. In *Advances in Cryptology — CRYPTO '98*, pages 1–12. Springer Berlin Heidelberg, 1998.

[16] Ernest F. Brickell, Jan Camenisch, and Liqun Chen. Direct anonymous attestation. In *ACM Conference on Computer and Communications Security, CCS*, 2004.

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

[18] Jan Camenisch and Anna Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In *Advances in Cryptology – CRYPTO 2004*, pages 56–72. Springer Berlin Heidelberg, 2004.

[19] Liqun Chen and Jiangtao Li. Flexible and scalable digital signatures in TPM 2.0. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security - CCS '13*. ACM Press, 2013.

[20] Liqun Chen, Dan Page, and Nigel P. Smart. On the design and implementation of an efficient DAA scheme. In *Lecture Notes in Computer Science*, pages 223–237. Springer Berlin Heidelberg, 2010.

[21] The RAINBOW Consortium. Rainbow reference architecture. Deliverable D1.2, 2020.

[22] The RAINBOW Consortium. Rainbow stakeholders requirements analysis. Deliverable D1.1, 2020.

[23] The RAINBOW Consortium. Rainbow attestation model and specification. Deliverable D2.1, January 2021.

[24] The RAINBOW Consortium. Rainbow use-cases descriptions. Deliverable D1.3, January 2021.

[25] Anupam Datta et al. A logic of secure systems and its application to trusted computing. In *30th IEEE Symposium on S&P*, pages 221–236. IEEE, 2009.

[26] Benjamin Edelman. Adverse Selection in Online 'Trust' Certifications and Search Results. In *Electronic Commerce Research and Applications 10*, pages 17–25, 2011.

[27] ETSI. Trust and Privacy Management, 2012. http://www.etsi.org/deliver/etsi_ts/102900_102999/102941/01.01.01_60/ts_102941v010101p.pdf [Online; accessed 26-August-2017].

[28] Intelligent Transport Systems (ITS); Security; Security Header and Certificate Formats. Technical specification, October 2017.

[29] Thanassis Giannetsos and Ioannis Krontiris. Securing V2X Communications for the Future: Can PKI Systems Offer the Answer? In *14th Int. ARES Conf*, 2019.

[30] Stylianos Gisdakis, Thanassis Giannetsos, and Panos Papadimitratos. SPPEAR: Security & Privacy-preserving Architecture for Participatory-sensing Applications. In *Proceedings of the 2014 ACM Conference on Security and Privacy in Wireless & Mobile Networks*, WiSec '14, pages 39–50, New York, NY, USA, 2014. ACM.

[31] Stylianos Gisdakis, Marcello Lagana, Thanassis Giannetsos, and Panos Papadimitratos. SEROSA: service oriented security architecture for vehicular communications. In *VNC*, pages 111–118. IEEE, 2013.

[32] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on computing*, 1989.

[33] Philippe Golle and Kurt Partridge. On the anonymity of home/work location pairs. In *Proceedings of the 7th International Conference on Pervasive Computing*, Pervasive '09, pages 390–397, Berlin, Heidelberg, 2009. Springer-Verlag.

[34] Infineon Technologies AG. *OPTIGA™ TPM SLB 9670 TPM2.0*, December 2018. Rev. 1.4.

[35] P. Jamshidi, C. Pahl, N. C. Mendonça, J. Lewis, and S. Tilkov. Microservices: The journey so far and challenges ahead. *IEEE Software*, 35(3):24–35, 2018.

[36] Vlastimil Klíma, Ondrej Pokorný, and Tomáš Rosa. Attacking RSA-based sessions in SSL/TLS. In *Lecture Notes in Computer Science*, pages 426–440. Springer Berlin Heidelberg, 2003.

[37] Nikos Mavrogiannopoulos, Frederik Vercauteren, Vesselin Velichkov, and Bart Preneel. A cross-protocol attack on the TLS protocol. In *Proceedings of the 2012 ACM conference on Computer and communications security - CCS '12*. ACM Press, 2012.

[38] Petar Maymounkov and David Mazières. Kademlia: A peer-to-peer information system based on the xor metric. In Peter Druschel, Frans Kaashoek, and Antony Rowstron, editors, *Peer-to-Peer Systems*, pages 53–65. Springer Berlin Heidelberg, 2002.

[39] Christopher Meyer and Jörg Schwenk. SoK: Lessons learned from SSL/TLS attacks. In *Information Security Applications*, pages 189–209. Springer International Publishing, 2014.

[40] Graeme Proudler, Liqun Chen, and Chris Dalton. *Trusted Computing Platforms*. Springer-Verlag GmbH, 2015.

[41] Eric Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446, August 2018.

[42] Mohamed Sabt, Mohammed Achemlal, and Abdelmadjid Bouabdallah. Trusted execution environment: what it is, and what it is not. In *2015 IEEE Trustcom*, pages 57–64. IEEE, 2015.

[43] Reiner Sailer et al. Design and Implementation of a TCG-based Integrity Measurement Architecture. In *USENIX Security symposium*, pages 223–238, 2004.

[44] TCG. Dice layering architecture. Rev. 0.19, July 2020.

[45] TCG. Tcg tss 2.0 enhanced system api (esapi) specification Vers. 1 Rev. 08, May 2020.

[46] TCG. Tcg tss 2.0 tpm command transmission interface (tcti) api specification Vers. 1.0 Rev. 18, January 2020.

[47] Ronald Toegl. *On Trusted Computing Interfaces*. PhD thesis, Graz University of Technology, 2013.

[48] David Wagner and Bruce Schneier. Analysis of the ssl 3.0 protocol. In *Proceedings of the 2nd Conference on Proceedings of the Second USENIX Workshop on Electronic Commerce - Volume 2*, WOEC'96, page 4, USA, 1996. USENIX Association.

[49] Samuel Weiser. *Enclave Security and Address-based Side Channels*. PhD thesis, 6 2020.

[50] Stephan Wesemeyer, Christopher J.P. Newton, Helen Treharne, Liqun Chen, Ralf Sasse, and Jorden Whitefield. Formal analysis and implementation of a TPM 2.0-based direct anonymous attestation scheme. In *Proceedings of the 15th ACM Asia Conference on Computer and Communications Security*. ACM, oct 2020.

[51] Jorden Whitefield et al. Privacy-enhanced capabilities for VANETs using direct anonymous attestation. In *IEEE Vehicular Networking Conference*.

[52] Zhang Xiong, Hao Sheng, WenGe Rong, and Dave E. Cooper. Intelligent transportation systems for smart cities: a progress review. *Science China Information Sciences*, 2012.