CYBERNETICA

JOCONDE D4.1 System Specification and Architecture

Technical report

Version 1.0

20.05.2025

D-16-522

Public

Project JOCONDE (Joint On-demand COmputation with No Data Exchange)
https://cros.ec.europa.eu/joconde

Project Managers:	Fabio Ricciato (Eurostat),
	Baldur Kubo (Cybernetica)
Authors (Cybernetica):	Kert Tali,
	Armin Daniel Kisand,
	Anton Zakatov,
	Riivo Talviste

Cybernetica AS, Mäealuse 2/1, 12618 Tallinn, Estonia.

E-mail: info@cyber.ee, Web: https://www.cyber.ee, Phone: +372 639 7991.

Date	Version	Description
24.02.2025	0.1	Advanced draft
06.05.2025	0.2	Revised draft
20.05.2025	1.0	Revised version for publication

Disclaimer

This document was prepared by Cybernetica AS as part of a procured project under Service Contract No. ESTAT 2023.0400.

The opinions expressed in this document are those of the authors. They do not purport to reflect the opinions, views, or official positions of the European Commission.

This is an intermediate project report. Individual parts of the document may be revised and changed in future version updates in the course of the project, as well as based on feedback received from expert readers. The final version of the report is planned to be released at the end of the project in March 2026.

We encourage readers to provide feedback. If you would like to suggest corrections or improvements, or simply ask for clarifications, we are happy to hear from you. Please contact us via email at joconde@cyber.ee. Your feedback can help to improve future versions of this document.

Table of Contents

1 Introduction

1.1 Document Purpose

This document is the first version of a technical deliverable intended to present the technical architecture specifications of the JOCONDE System (System). The report will go through three subsequent versions – D4.1 (this document), D4.2 (revised version) and D4.3 (final version) – following an iterative development approach, each version representing a separate milestone. Subsequent versions will carry over the content of the previous ones, possibly with amendments, and add new material. Hence we advise the readers to always refer to the latest version available on the project website[1].

This document builds upon the previous JOCONDE project deliverables *D1.1 Usage Scenarios and System Requirements* [1] and *D2.1 Technology Survey and Analysis* [2]. The goal of this work is to enable subsequent interdisciplinary analyses and prototyping. As such, this document outlines a technical solution for the purpose of applying further domain knowledge – refining the business processes, organisational and governance models, and conducting risk and legal analysis – eliciting new requirements and feeding back into future iterations.

1.2 Intended Audience

This document is addressed to a technical audience spanning diverse specialisation domains. Security specialists should be able to assess additional unidentified technical risks, if any. Data protection specialists can benefit from information flow to understand how the System handles sensitive data. Business analysts should find this document useful to understand the technical safeguards that necessitate supplementary business processes. Prospective adopters from the statistics domain (e.g. NSIs and other organisations) should be able to use the context and deployment models found herein to assess how the proposed architecture is implemented from the IT and administrative standpoint. Vendors of MPC technologies may find the functional model useful for the assessment of integration opportunities with the System.

Before reading the present document, we recommend readers to go through the previous deliverable *D1.1 Usage Scenarios and System Requirements* [1] which establishes the core concepts on which the present document builds upon, including the notions of Planes, roles, business rules, and requirements.

1.3 Document Scope

The scope of the present document version was determined based on the following goals:

- Establish a foundational architecture;
- Focus on the core functionalities to be offered by the System;
- Provide sufficient detail to facilitate initial prototyping.

The foundational architecture is a baseline structure capturing requirements with the greatest architectural significance and risk, allowing requirements with lower risk to be addressed in later design stages. In conformance with the core values of the System as described in the vision document [1], the foundational architecture is based on ensuring security and usability. The scope

[1] JOCONDE Project https://cros.ec.europa.eu/joconde

of the document in terms of decisions, coverage of requirements, and level of detail has been formulated to take account of these aspects, leaving other aspects (e.g. evolution, development, operations, regulation) for future versions of the document.

The scope includes the core workflow functionality, describing the System in relation to user interactions on the Control Plane and Data Plane. We also detail the technological safeguards and measures that are required to protect Restricted Data throughout its lifecycle in the System.

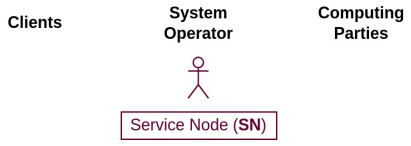
The defined scope exludes a detailed specification of the Management Plane; however, we do outline its positioning and role in the architecture. Requirements associated to the Management Plane are expected to be clarified based on new context derived from this document.

The document does not explicitly present new business processes.

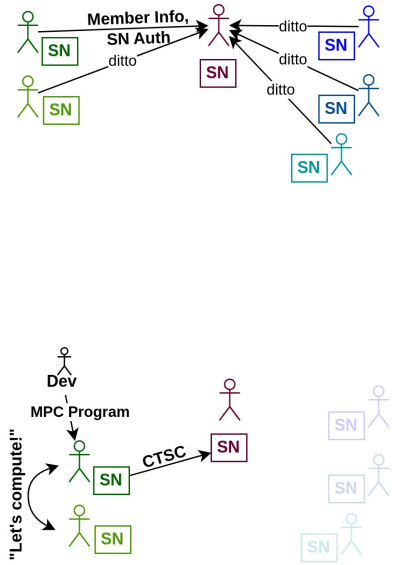
1.4 Overview

1.4.1 10,000 Foot View

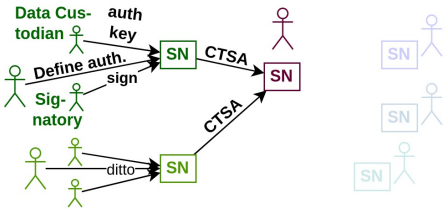
The JOCONDE system is a platform for coordinating, agreeing upon and executing Computation Tasks (CTs) without exposing the input data to any other entity other than their respective data holders. The goal of JOCONDE is to provide a ready-to-use tool to organisations that hold confidential data and want to maintain direct fine-grain control over what CTs are performed over their data in combination with other organisations' data.
Throughout the entire lifetime of each CTs, the data are protected in transit, storage, and processing through a combination of Secure Multiparty Computation (MPC) and Trusted Execution Environments (TEEs).

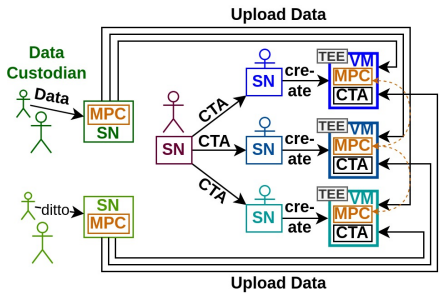


To this end, the JOCONDE system brings together Input Parties (IPs), Output Parties (OPs) and Computing Parties (CPs) on a ready-to-use platform which is operated by the System Operator (SO).
IPs and OPs represent the Clients, i.e. the entities that are served by System. CPs provide their capabilities to the Sytem in order for it to serve the Clients. Clients and CPs can join and leave the platform at any time. Newcomer Clients and CPs are required to install a Service Node on their (cloud) infrastructure, and exchange various legal and authentication information with the SO (onboarding procedure) before getting involved in any CTs. The Service Nodes form a tree structure with the SO at the root. Larger clients, i.e. companies, may want to add further internal Service Nodes, thereby increasing the depth of the tree.



A group of Clients that decide to perform a joint computation become Peers for that specific CT and they must all sign the corresponding Computation Task Specification Core (CTSC). The CTSC specifies the exact MPC program (or Algorithm) to be executed and the permissions of each Peer, among other details for the CT at hand. While initial coordination is assumed to be performed outside of the System, all further steps take place through the web interface of the Peer's Service Node.
The Task Organiser assembles the CTSC and sends it to the SO. Then every Peer reviews the CTSC and adds the Computation Task Specification Addendum (CTSA). This addendum refines the authentication and authorisation information: they grant individual Data Custodians and Data Analysts permissions to submit specific input data or retrieve specific output data, respectively. A Signatory (known to the SO) legally signs the addendum.





When the SO has approved all the CTSAs they send the resulting Computation Task Agreement (CTA) to the CPs. The latter create dedicated VM-based TEEs to protect the input data submission, the execution of the MPC program(s), the cross-VM communication, and the output data retrieval. Data Custodians (and, subsequently, Data Analysts) use their respective Service Nodes to talk directly to the ephemeral VMs.

1.4.2 Architectural Design Approach

This document adopts the *viewpoints and perspectives* approach[2] as described by Rozanski and Woods [3]. This approach is based on the idea that a multi-faceted architecture can be best understood by considering it as a collection of different structural aspects, called views. A viewpoint is a template specifying a set of important elements to consider for a particular view. The core viewpoints are: context, functional, information, concurrency, deployment, development, and operational viewpoint. The following views are presented in the current version of the document:

Context View
: Establishment of an abstract governance model that avoids centralised trust; identification of external and internal actors, and listing their interaction scenarios

Deployment View
: Capturing the physical placement of components and services within different organisational boundaries; specification of access points and connection paths

Functional View
: Identification of functional elements incorporating the necessary degree of automation; description of security-critical components and constraints

Information View
: Presentation of the data model and information flow model that provide integrity assurances and trust between parties

Subsequent versions will add new views and supplement existing ones.

The different views are helpful for breaking down the model and presenting its structure in a more manageable form. However, the views by themselves are often insufficient to convey the bigger picture of how the system fulfils the requirements. The role of *perspectives* is to show how the elements described by different views combine to fulfil the system's requirements. Perspectives address groups of interrelated issues, e.g. accessibility, usability, performance, regulation, security. In this version, we focus on the security and usability perspectives.

1.4.3 Document Structure

The remaining part of the document is structured into four chapters. Chapter 2 introduces the key stakeholders and stakeholder groups, and the main system requirements. Chapter 3 describes the *architectural forces*, i.e. the driving factors that have shaped the architecture. Chapter 4 presents the four architectural views: context view, deployment view, functional view, and information view. Chapter 5 presents the security and usability perspectives applied to views listed above.

Appendix A documents the significant architectural design decisions and their rationale. Appendix B shows examples of data structures as code.

[2] Software Systems Architecture https://www.viewpoints-and-perspectives.info/ Last accessed: February 2025.

2 System Stakeholders and Requirements

This chapter describes the stakeholders and requirements that informed (or are addressed by) the current architectural design. The full listing of stakeholders and requirements can be found in *D1.1 Usage Scenarios and System Requirements* [1], with which the following information remains consistent, adding detail where necessary.

2.1 Stakeholders

D1.1 Usage Scenarios and System Requirements, Section 2.3 identified several classes of *organisation stakeholders* which are important in this architecture document to understand the organisational context, cross-organisational trust, technical capabilities, and their inner stakeholders and structure. *Class of* stakeholder implies an abstract entity, which may be embodied by one or several specific stakeholders.

Member
: Is an organisation (legal person) that has an identity in the System and has been approved by the SO to interact with it. The sub-classes of a Member are Client and CP.

  Client
  : Is a member of the ESS (e.g. NSI or other national authority) or partner with ESS members (e.g. a private or public data provider or research organisation). The Clients are served by the System. A Client (legal person) has one or more Users (natural persons).

  Computing Party (CP)
  : Supplies the System with computation resources.

System Operator (SO)
: The SO runs the System and provides support to Members. In the scope of this project we assume that Eurostat takes the the role of SO.

In this document we present classes of individual stakeholders to describe the System's interactions in more detail.

Task Organiser
: An employee of the Client, likely a project manager, whose role is to oversee the successful consolidation of CTs and act as the point of contact for the CTs.

Data Custodian
: An employee of the Client who is authorised to access the Restricted Data of the organisation. They supply the Input Data to CTs.

Data Analyst
: An employee of the Client who processes and interprets computation results. They retrieve the Output Data.

Signatory
: An authorised signatory of the Client organisation.

2.2 Requirements

This section lists requirements which had a significant role in shaping the architecture in this iteration. The following requirements were introduced in document *D1.1 Usage Scenarios and System Requirements* [1]. Some of the listed requirements have been revised to improve clarity.

BUS-1.4 ***Requirement*** *(System roles)*
Clients shall be able to initiate new Computation Tasks with minimal manual intervention by other entities such as the System Operator or Computing Party to streamline operations and lessen personnel workload.

BUS-1.5 ***Requirement*** *(System roles)*
Clients can invite other Clients to take part in their Computation Task as Input or Output Parties.

BUS-1.6 ***Requirement*** *(System roles)*
All Input and Output Parties must accept and approve the details of the planned statistical computation as defined in the Computation Task Agreement, and acknowledge any residual risk, before the Computation Task is executed.

BUS-2.4 ***Requirement*** *(Governance roles)*
The System Operator operates the System components responsible for collaboration and coordination mechanisms in a manner which, within the scope of the considered attack model, rules out any possibility of attacks against System Members and their data.

BUS-2.8 ***Requirement*** *(Governance roles)*
The System Operator shall not represent a single point of trust.

BUS-4.2 ***Requirement*** *(Policies)*
No single entity shall have the possibility to reveal the Protected Data (Input, Interim, or Output Data), unless explicitly stated in the Computation Task Agreement.

BUS-4.4 ***Requirement*** *(Policies)*
A Computation Task shall be executed only after approval by all Input and Output Parties.

BUS-4.9 ***Requirement*** *(Policies)*
Protected Input Data and Interim Data connected to a Computation Task shall be securely deleted or rendered permanently illegible once the Computation finishes or the deadline is reached.

BUS-5.1 ***Requirement*** *(Supplementary)*
All operations within the System shall be as simple and lightweight as possible for the Clients.

SYS-1.7 ***Requirement*** *(System security)*
The Computing Nodes shall employ secure hardware technologies, e.g. TEE with hardware isolation.

SYS-1.10 ***Requirement*** *(System security)*
The System should use a combination of multiple technologies and security/privacy layers with complementary security guarantees to achieve the highest possible degree of protection and trustworthiness.

SYS-2.1 ***Requirement*** *(Trust)*
The System shall enable Users to verify the identities of other Members contained in the Computation Task Specification locally. The verification shall not rely on trust in the System Operator.

SYS-2.2 ***Requirement*** *(Trust)*
The System shall use a Computation Task Specification approval mechanism that provides integrity and non-repudiation.

SYS-2.3 ***Requirement*** *(Trust)*
The System shall allow Input and Output parties to verify the integrity of the copies of their Computation Task Agreements deployed in Computing Nodes.

SYS-3.1 ***Requirement*** *(Using the System)*
Preparation, configuration, and execution of a Computation Task in the System should be as simple and lightweight as possible for the Clients and should involve only minimal marginal costs.

SYS-3.4 ***Requirement*** *(Using the System)*
The System shall support Input Parties and/or Output Parties taking the role of a Computing Party in Computation Tasks which they are a part of.

SYS-4.1 ***Requirement*** *(Managing the System)*
The System shall provide the System Operator with Identity and Access Management (IAM) for Member management and access.

SYS-5.2 ***Requirement*** *(System)*
Computing Nodes shall only accept and enforce Computation Task Agreements sourced by trusted means.

SYS-5.3 ***Requirement*** *(System)*
The System shall maintain a log of the proceedings of each Computation Task, containing Computation Task lifecycle events, and make it available to all Parties of the Computation Task.

SYS-6.1 ***Requirement*** *(Task and data lifecycle)*
The System shall associate all data with its respective Computation Task to enforce data lifecycle policies.

SYS-6.5 ***Requirement*** *(Task and data lifecycle)*
The System shall employ technical measures ensuring that all Protected Input Data and Interim Data are permanently and securely deleted or rendered permanently illegible immediately after the completion of the Computation Task or upon reaching the deadline.

SYS-6.6 ***Requirement*** *(Task and data lifecycle)*
The System shall employ technical measures ensuring that all Protected Output Data are permanently and securely deleted or rendered permanently illegible immediately after retrieval by all Output Parties or upon reaching the deadline.

SYS-7.4 ***Requirement*** *(Computational capabilities)*
The System should allow Members to choose the MPC technology for each Computation Task and specify the chosen technology in the Computation Task Specification.

SYS-9.3 ***Requirement*** *(Privacy)*
The System shall not execute any computing functions other than the ones approved in the Computation Task Agreement.

3 Architectural Forces

3.1 Goals

Design goals are formulated in accordance with *D1.1 Usage Scenarios and System Requirements* [1] which establishes the business drivers of the Project. The main goal of the Project is to design a System which makes it easier to use SPC technologies in the official statistics production process to protect against unauthorised access to Restricted Data. More specifically, the aim is to create a self-service capability enabling a large number of Parties to leverage a single MPC deployment for their CTs.

The System must strike a balance between high security and ease of use. On the one hand it must integrate the use of MPC and TEE technologies, taking care that these technologies are utilised correctly. The technical solution wrapping these technologies should not negate the offered security guarantees. On the other hand it should make the fewest possible compromises in usability, enabling non-technical users to benefit from the system.

> While the technical model provides the functional tools for the achievement of specific business goals, effective utilisation of the model hinges on understanding and adhering to correct procedures. These procedures often involve processes which are outside the scope of this document: more specifically, interactions between external actors extending beyond the technical domain.
>
> As the System enables the use of privacy-enhancing technologies for general-purpose statistical production, the efficacy of the security measures may potentially be diminished either knowingly or by accidental misuse. Achievement of an appropriate level of protection requires a holistic understanding of both the technical capabilities of the described tools and the broader operational environment which they serve. This document elaborates on the technological safeguards for the mitigation of known risks. These safeguards, presented here in the nature of *what can be achieved technologically*, must be contextualised by incorporating them into the sociotechnical context.
>
> The close relationship between technical and sociotechnical models is illustrated by Figure 1.

3.2 Constraints

The business requirements and the chosen SPC technologies themselves impose the following limitations on architectural choices:

- The System must make use of both MPC and TEE technologies at Computing Nodes (SYS-1.1, SYS-1.6, SYS-1.7), with at least three CPs for MPC (SYS-5.1);
- The System must remain secure under the collusion of, or intrusion into, two out of three CPs (SYS-1.2, SYS-1.3);
- The System supports several MPC Engines in parallel (BUS-5.4);
- The System must not have a single point of trust, i.e. no single Party should be able to access Input or Output Data or alter the Computation Task lifecycle unless this is explicitly agreed upon by the relevant Parties in advance;
- Using the System must not require the use of specialised hardware from Clients (IPs and OPs) (SYS-3.2).

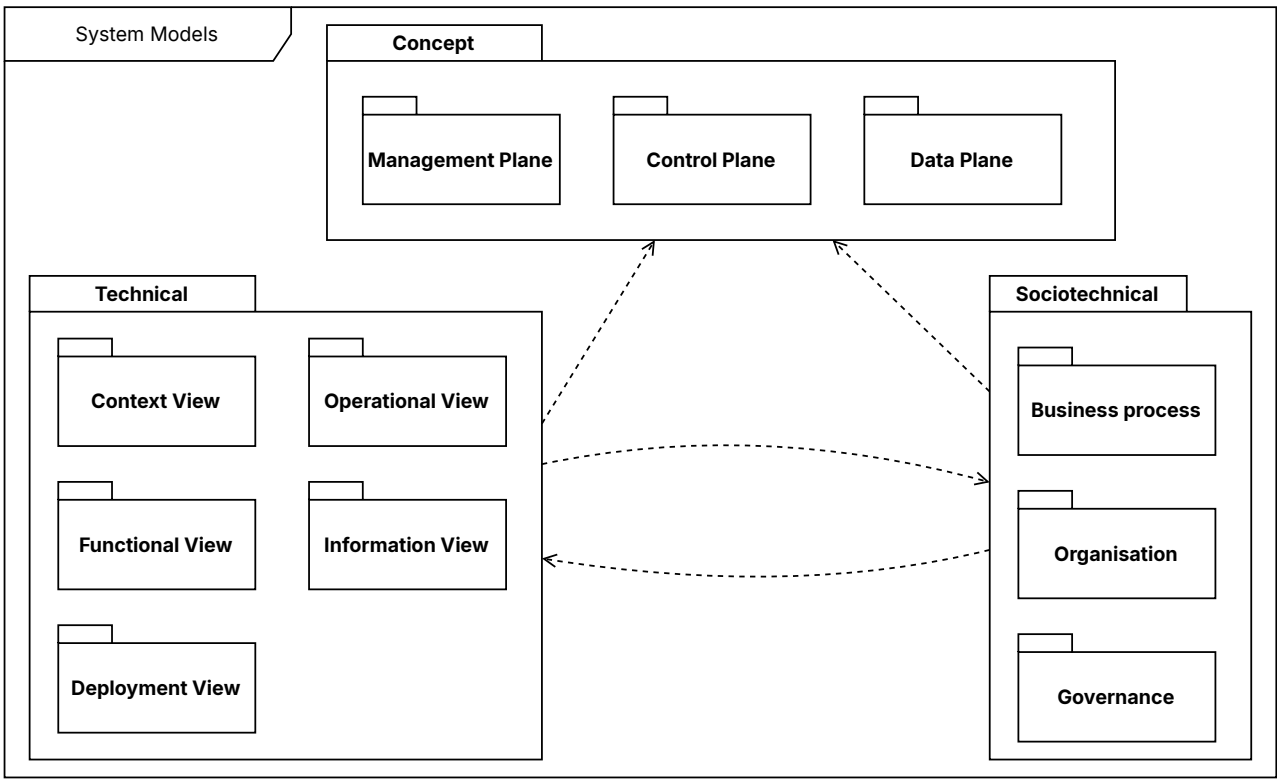


Figure 1. Relationship between technical and sociotechnical models

3.3 Architectural Principles

Given the goals and constraints described above, the architecture is designed in accordance with a set of principles. In the following list we have formulated a number of overarching principles. Individual principles are referred to in relevant architectural decisions to foster consistency.

Plane separation (PR-1)

The System should exhibit a logical and functional separation of responsibilities w.r.t the conceptual Planes. This reinforces the overarching model, tying together processes with interactions and functional elements. Plane separation would guide the specification of inter-Plane interfaces, making sure to achieve the suitable level of decoupling.

Decentralised governance (PR-2)

On the top level, the JOCONDE System identifies three classes of organisational stakeholders: the SO managing the System, Clients using it, and the CPs providing the computing resources. In *D1.1 Usage Scenarios and System Requirements* [1], each of these stakeholders were considered as legal persons which are joined to the System through a one-time onboarding procedure. As the System is intended to be used and operated by individuals from those organisations, we can assume that the actual identities of those individuals is irrelevant in any inter-organisational interactions. Intra-organisationally, however, the distinction is important, as the organisation must have control over who is allowed to interact with the System on their behalf. The onboarding procedure naturally delegates select management and administrative duties to these organisations.

From the System's perspective, this assumption is an opportunity to go one step further and adopt a decentralised governance model. This means giving organisations full autonomy over their domain by technological means, benefitting trust and transparency while lowering the management burden of the SO.

No Single Point of Trust (PR-3)

The JOCONDE System aims to be the alternative solution for the *status quo* of computing by means of a trusted third party. One of the core principles of MPC is that an MPC system can not be under the full control of a single trusted entity, but rather exhibit distributed control over the computation among a set of parties that are trusted collectively. This is also applies to systems building on top of MPC, or using MPC as a tool. It is critical that the System is designed around the MPC paradigm, ensuring the confidentiality of Restricted Data and the integrity of the computed function. The notion of avoiding a Single Point of Trust (SPoT) is attributed to a wide range of patterns and technological safeguards described in this document.

MPC as a Service (PR-4)

As expressed in the vision [1], the JOCONDE System should be reminicent of the SaaS model from the point of view of its Users in achieving their goal of running general data analysis under the MPC paradigm. This goal represents two stages of the core workflow described in *D1.1 Usage Scenarios and System Requirements, Section 4.1*: (1) the secure agreement enabled by the SO, followed by (2) the secure computation enabled by the CPs. These stages would ideally be transparent to the user. This principle combines two architecturally significant considerations:

Multi-tenant and shared infrastructure – components which offer these services must account for servicing many concurrent contexts in isolation;

On-demand and self-service – which internal processes require automation to offer a streamlined user experience and minimise manual intervention.

We refer to a system of this nature as an MPC as a Service (MPCaaS). The MPCaaS model is a generalisation of the JOCONDE System, which is tailored specifically for the production of official statistics.

Technology integration strategy (PR-5)

The System should accommodate for the evolution of the underlying secure computation technologies. It should be clear what the System considers as a suitable MPC or TEE technology in terms of behaviour, capabilities, environmental factors, and high-level interfaces. In designing the interfaces between these components and the System, the specification shall consider the state of the art of several existing candidates to settle on a reasonable set of requirements.

Data plane modularity (PR-6)

The System considers secure computation technologies as external components. Each TEE and MPC technology should be encapsulated within its own module. New technologies (or integrations thereof) should be simply pluggable into the System without neccessitating modifications to other components. While the process of adding new modules does not need to be supported at runtime (but rather depend on the deployment configuration) the users should be able to choose from available modules on a per-CT basis. Thus, the System can serve a variety of different computational capabilities.

4 Architectural Views

4.1 Context View

The System conceptually spans organisational boundaries of multiple entities. This is apparent from several requirements[1]. In more specific terms, we are describing a distributed system in which different Members are in charge of operating the components particular to their role(s) in conjunction with the SO. They do so by means of their own infrastructure, enabling a high level of control over System functions. Hence, while the SO assumes governance of the whole System on the business level, the technological side of it is jointly governed. The System may grow and shrink during its lifetime following the onboarding and offboarding of Members.

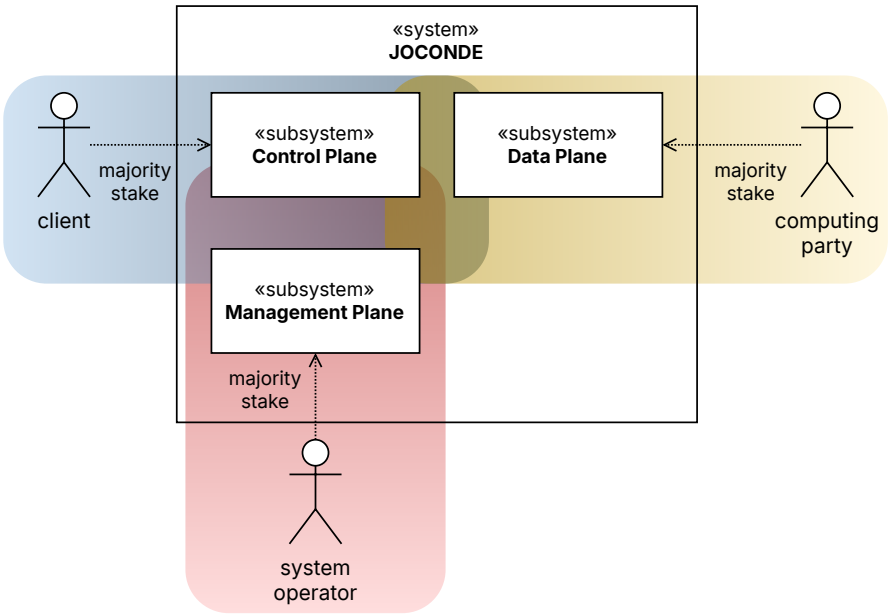


Figure 2. Mapping of roles to subsystems – coloured areas signify organisational boundaries; the boundary and subsystem overlap represents stake in operating components within that subsystem

The use of Planes for the description of the System context is motivated by the plane separation principle (PR-1), while the association of the planes with governance is rooted in the decentralised governance principle (PR-2) and no SPoT principle (PR-3). Figure 2 illustrates the separation of organisational domains and how these are distributed to govern the different subsystems.

[1] See BUS-2.4, BUS-4.7, BUS-4.8, SYS-5.1 in *D1.1 Usage Scenarios and System Requirements* [1].

> The dynamic make-up of the System would imply that it could exist in a partial state if there are not enough participants to assume the defined roles. In such case the System would not be able to serve its intended function. Hence, when describing the context of the System, we assume a *fully operational* state. The following minimal conditions have to hold:
>
> 1. The SO has deployed their Management Plane and Control Plane components.
> 2. At least three CPs have deployed their Control Plane and Data Plane components, and are interfaced with the System.
> 3. At least two Clients have deployed their Control Plane components, and are interfaced with the System.
> 4. The System incorporates at least one MPC technology and at least one TEE technology.

External and internal entities. Entities are considered *internal* if they are parts of the System's internal structure. Even though the number of active Clients and CPs is dynamic, their contributed infrastructure and services are nonetheless regarded as internal to the System. Additionally, internal entities include actors such as IT Administrators, who operate (parts of) the System.

External entities are:

1. Actors using the System;
2. Technological components which augment the System, but are independently developed, e.g., MPC Engines;
3. External services that the System depends on, e.g., the EU/EEA Trust List.

Figure 3 illustrates the system context.

- The System is jointly operated by a number of organisations, each contributing infrastructure and personnel. The reasoning behind this is presented in Decision ADR-2.
- The System integrates a number of secure computation technologies supporting its computational capabilities.
- Client-specific actors interact with the System strictly via the Client Node located inside their organisation. The same applies to SO-specific actors.
- The internal IT Administrators handle non-specific operations, e.g. infrastructure maintenance and configuration, which are distinct from interactions.

4.1.1 Interaction Scenarios

D1.1 Usage Scenarios and System Requirements [1] emphasises ease of use as a critical aspect of using the core functionality of the System. Most of all, this concerns the Data Custodian, Task Organiser, and Signatory as the primary users. Their interactions with the System are centred around the Control Plane and Data Plane. Figures 4 and 5 respectively present simplified overviews of the use cases on each plane.

> The illustrated use cases represent the functionalities covered in greatest detail in the design of the current iteration. We account for the fact that user interactions are also necessary in other aspects of the System, e.g. management and auditing, the extent of which requires further analysis before being formalised. Future iterations of this document will refine and supplement the use cases, taking into consideration the initial System design paired with a business process solution.

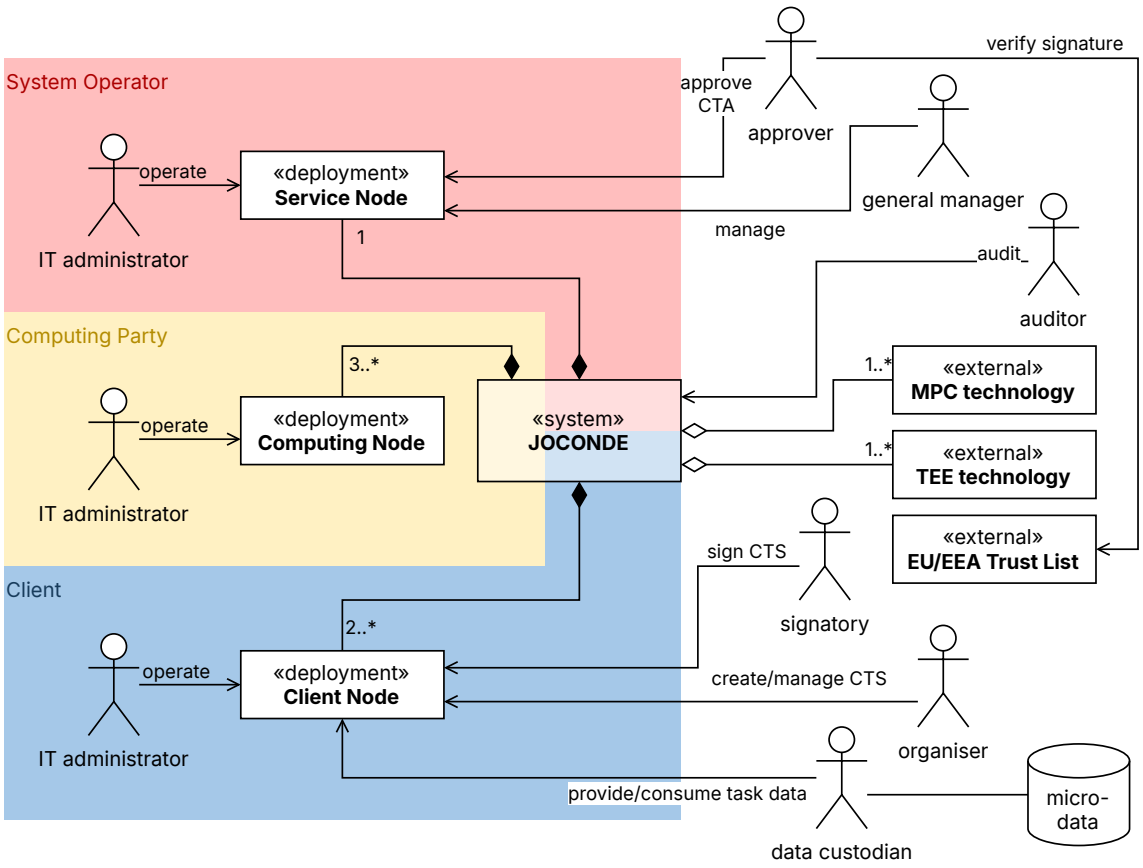


Figure 3. The context model of the System, with the three coloured sections marking the organisational boundaries of the specified roles. Internal entities are those shown within the organisational boundaries

4.1.1.1 Control Plane user interactions

We describe the user interactions associated with the Control Plane based on the use cases described in Figure 4.

Manage CTS

- Covers all of the Task Organiser's activities in the consolidation of a CT in collaboration with other Clients' Task Organisers.
- Involves the Task Organiser logging in to their organisation's Client Node if applicable[2].
- All interactions are facilitated by a web-based user interface.

Initiate CTS

- Done by the Task Organiser among the IPs and OPs with initiative to specify the core details of the Computation Task Specification (CTS).
- Accomplished by filling out a web form, as shown in Figure 6.
- Marks the creation of a new CTSC in the System.

Assign users to CTS

- Done by each Task Organiser of the IPs and OPs who takes part in the CT.

[2]Depending on the Client's environment, authentication may be unnecessary. See Section 4.2.

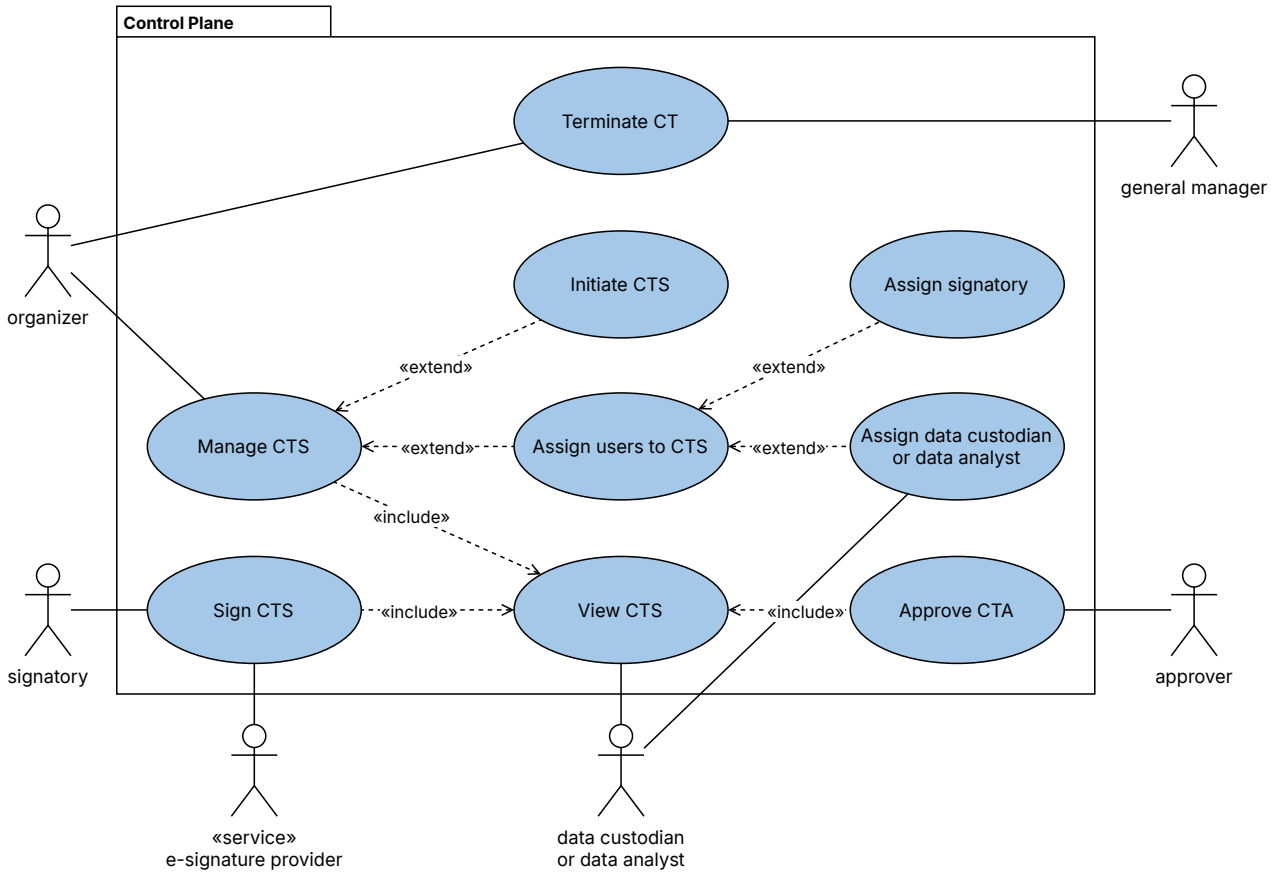


Figure 4. Use case diagram of high level interactions on the Control Plane

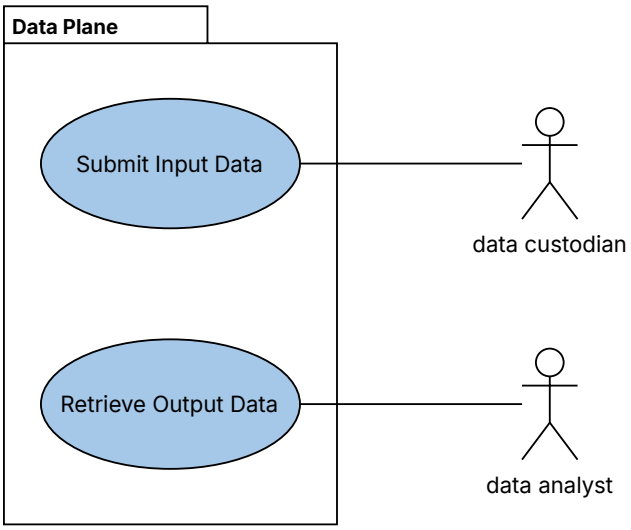


Figure 5. Use case diagram of high level interactions on the Data Plane

- Allows the Task Organiser to delegate roles to personnel inside their organisation within the scope of the CT.
- This includes specifying the Signatory who will sign the CTS on behalf of the Client, or the Data Custodian or Data Analyst who will upload input and download output.
- Marks the creation of a CTSA, as shown in Figure 8.

View CTS

- Any User with access to the service can get an overview of CTSs they are assigned to, as shown in Figure 7.
- This enables Users to inspect the CTS contents and review the actions required by them.
- Further, they can monitor the progress of the CTS consolidation and CT execution.

Sign CTS

- If applicable, the Signatory of the Client organisation signs the CTS with a legally binding signature.
- This is illustrated in Figure 9.

Approve CTA

- If applicable, the Approver reviews CTAs to green-light them for execution.

Terminate CT

- General Managers and Task Organisers both have the ability to call off the execution of an in-progress CT.

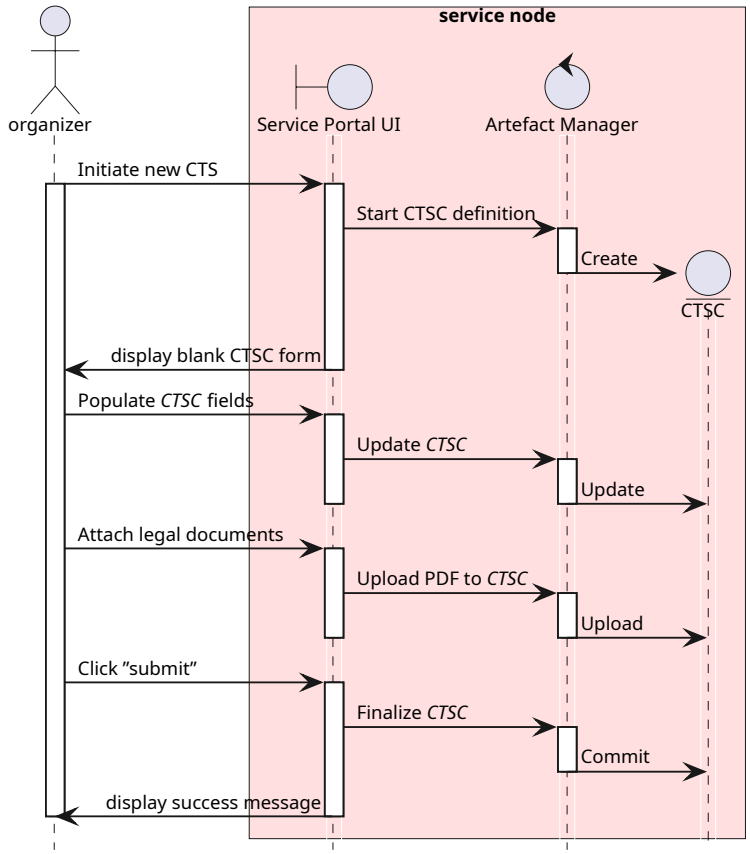


Figure 6. Use case: Initiate CTS

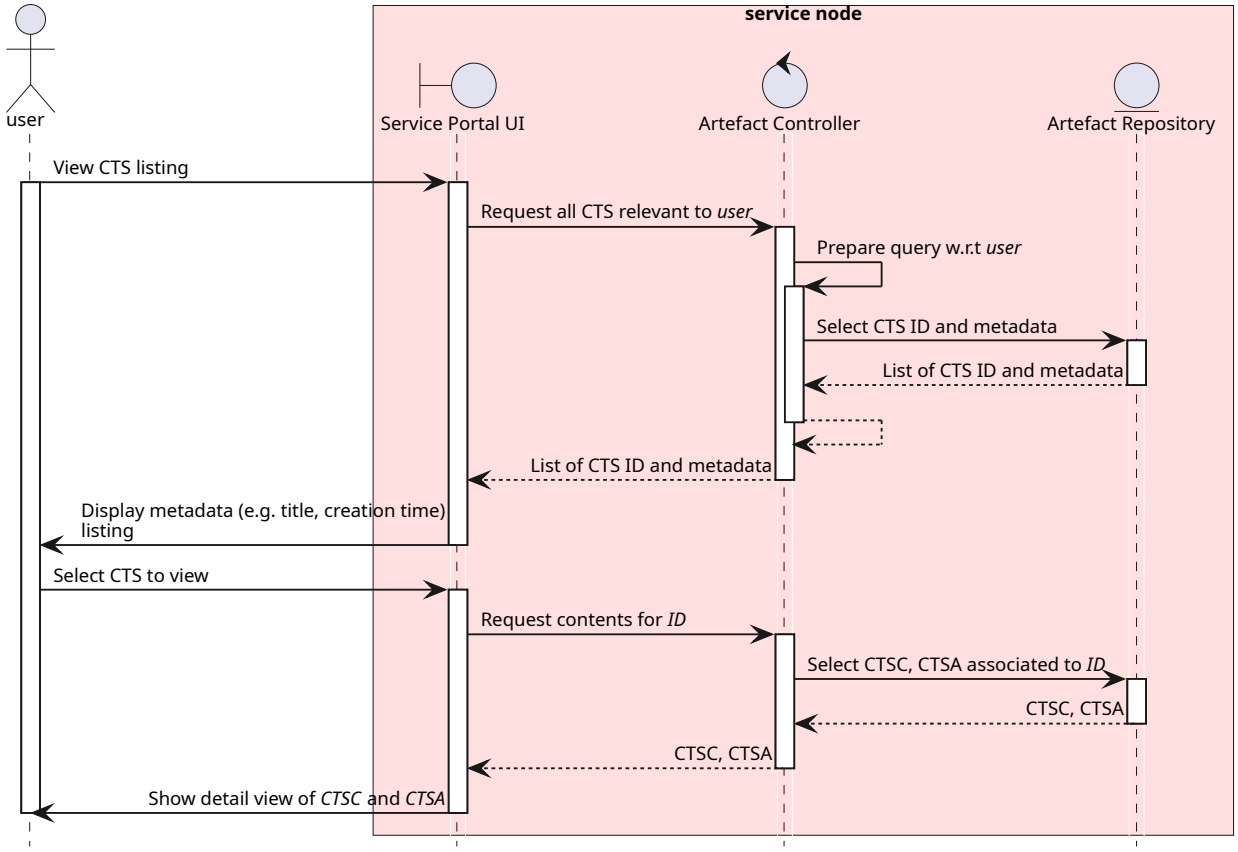


Figure 7. Use case: View CTS

4.1.1.2 Data Plane user interactions

We describe the user interactions associated with the Data Plane based on the use cases described in Figure 5.

Submit Input Data

- The Data Custodian uses a web-based data submission/retrieval interface for supplying a CT with Input Data before its execution.
- They select the intended CT and data table to supply.
- They load a data file from disk. The file must conform to the formatting and schema requirements specified in the CTS.
- The System returns a report of successful upload or failure along with the reason.

Retrieve Output Data

- The Data Analyst uses a web-based data submission/retrieval interface after the CT has finished.
- They select the intended CT and Output Data elements to download.
- They save the resulting data file(s) to disk.

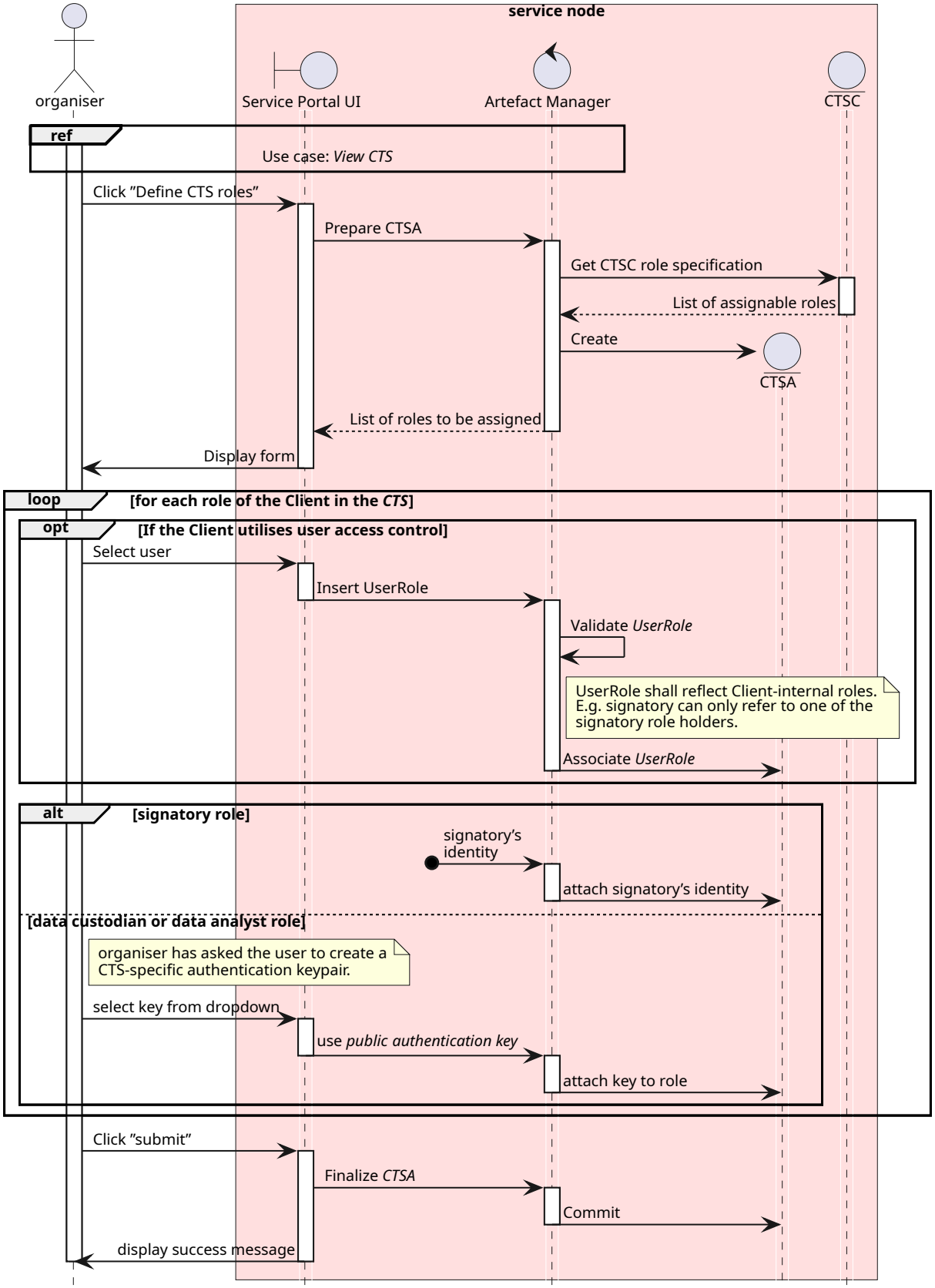


Figure 8. Use case: Assign users to CTS. This sequence diagram only shows the Task Organiser's interactions, whereas assigning a Data Custodian or Data Analyst requires their collaboration in creating a Public Authentication Key (PAK) as shown in Figure 35

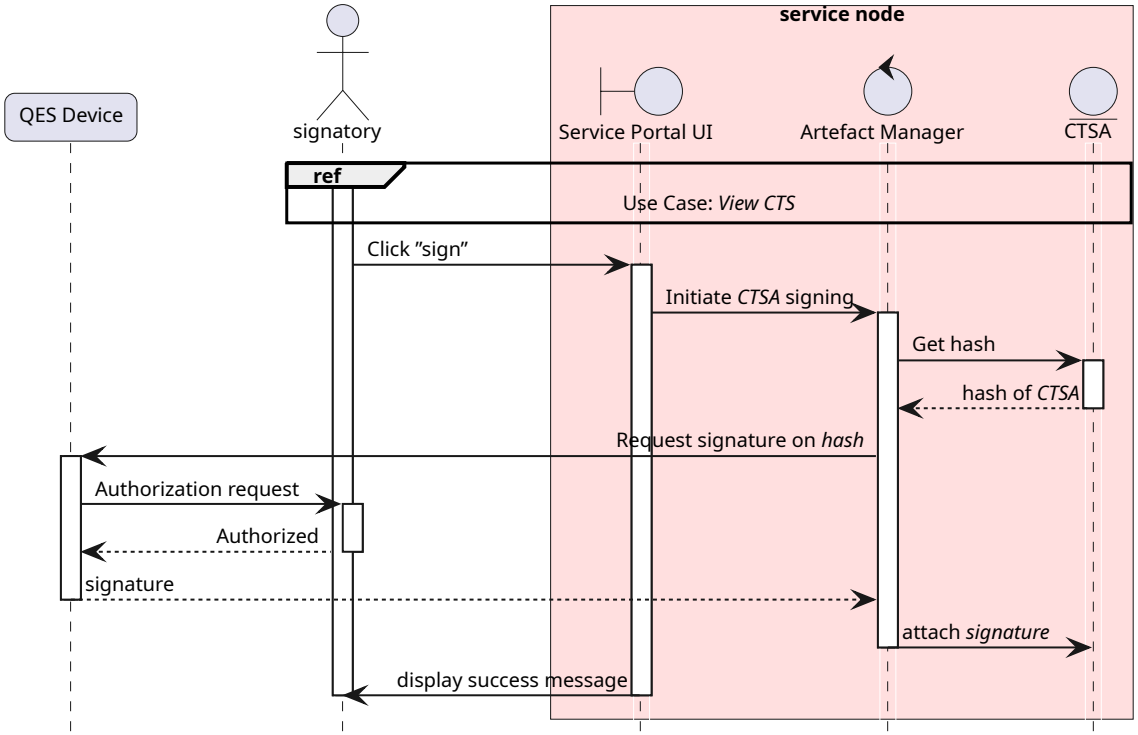


Figure 9. Use case: Sign CTS

4.2 Deployment View

This section concerns the physical makeup of the System. The following view explains some of the most important design decisions that play significant roles in the subsequent views.

4.2.1 Runtime platform composition

The deployment model is most influenced by the security design goals outlined in *D1.1 Usage Scenarios and System Requirements, Section 3.3*. It is clear that a deployment calls for at least **three independent Computing Nodes** as a prerequisite for the utilisation of MPC. Consequently, the System has to account for the **secure transmission of Restricted Data** between the Data Custodians and the Computing Nodes. The common approach is to facilitate the protection of data at the edge (on-premises of the IP) to avoid moving unprotected data and increasing the attack surface, implying a **client-side deployment**. Furthermore, the deployment model must express the **positioning of TEEs** to benefit protection and trustworthiness.

Outside of security requirements, it is imperative for the System to enable some degree of centralised control (by the SO) and visibility (by the SO and System Auditor (SA)). Alongside the MPCaaS principle (PR-4), a fully decentralised approach would be unnecessarily complex. Thus the System incorporates a **central element** in its deployment for facilitating Control Plane and Management Plane functions. This reverts attention back to the risks associated with having central control which, without due regard, could violate the no SPoT principle (PR-3).

The reasoning behind Decision ADR-2 highlights the significance of *where the software is deployed* for ensuring truly trustworthy interactions for the Users. It implies that a client-side deployment is necessitated not only by Data Plane interactions (see Figure 5) but also by several security-critical aspects[3] of the Control Plane. Each Client would therefore need to host at least one ***node***

[3] See *D1.1 Usage Scenarios and System Requirements, Section 3.3, paragraph 'Security measures in the Control Plane'*

that mediates *services* between the wider System and its local user base.

Below, we use Service Node as a general term for most of the deployment targets, including the SO's central server and any of the Member's nodes connected to the server. This arrangement has a star-shaped topology, but since the child Service Nodes are free to define their own hierarchies, the structure can more accurately be described as a *tree topology*. Decisions ADR-3 and ADR-4 provide further insight into the reasoning behind these abstractions. Figure 10 shows an example of a System deployment arrangement (the three-dimensional blocks) accounting for all the information presented in this section so far.

4.2.1.1 Service nodes

Each Member operates at least one Service Node as the service gateway for their organisation but otherwise possesses complete autonomy over their internal infrastructure. They are free to adapt their internal service node network to align with their organisational structure, ranging from minimal deployments to complex departmentalised setups. The former may omit any distinction of roles, access control, or policies, befit for a one-man operation (or one-off participation). The latter is more suitable for invested participants who are seen as using the System over a longer period for many different CTs, as well as large organisations who need to separate concerns over personnel or groups thereof. Flexibility enables the System to align with the ease of use requirement, but also recognises that a Member stakeholder's technological capability, requirements, and internal structure are indeterminate. To facilitate the deployment of a Service Node at the site of a Member, the software vendor should provide adequate deployment automation scripts[4] and documentation, accounting for technological prowess and the nature of underlying infrastructure.

The Service Nodes exchange information over Machine-to-Machine interfaces, with the connections between them secured by mTLS using cryptographic keypairs which are established during the onboarding procedure (see Section 4.4). Thus, the minimum criteria for a Member's deployment are adherence to the established communication protocols at their Service Node gateway and respective rules in the corporate firewall , ensuring interoperability with the SO Service Node.

> Note that a Service Node is a dynamic structure. To support the narrative of the examples presented here, certain Service Nodes have been given specific designations describing their roles and capabilities. In reality, these are determined by deployment and runtime configuration. As an illustration, take Requirement SYS-3.4, which calls for the Clients to be able to serve as CPs in select CTs. According to the business process [1], the organisation in the Client role must also onboard the System in the CP role. From the operational perspective, however, they must only augment their existing Service Node with the respective software components that handle the running of the CTs, while the SO adjusts their policy to permit CTs to be executed by the given node.

Figure 10 shows Service Nodes containing components organised into packages signifying the software they host (i.e. their capabilities). The contents of the packages are shown in Figure 11. Here we will only provide a high-level overview of the capabilities and responsibilities based on the components, leaving detailed definitions of the components for Section 4.3.

[4] In Figure 10 we have labeled the Service Node with the «container» stereotype, signifying that the preferred method of deploying System software makes use of containerisation technologies. This fosters enhanced portability and operational workflows (updating and configuration) of deployments in heterogeneous environments. The figure is a simplification in the sense that realistically, the components within the Service Node would manifest multiple containers for different sets of capabilities, orchestrated using standard tools like *Docker Compose* or *Kubernetes* that provide the aspect of automation.

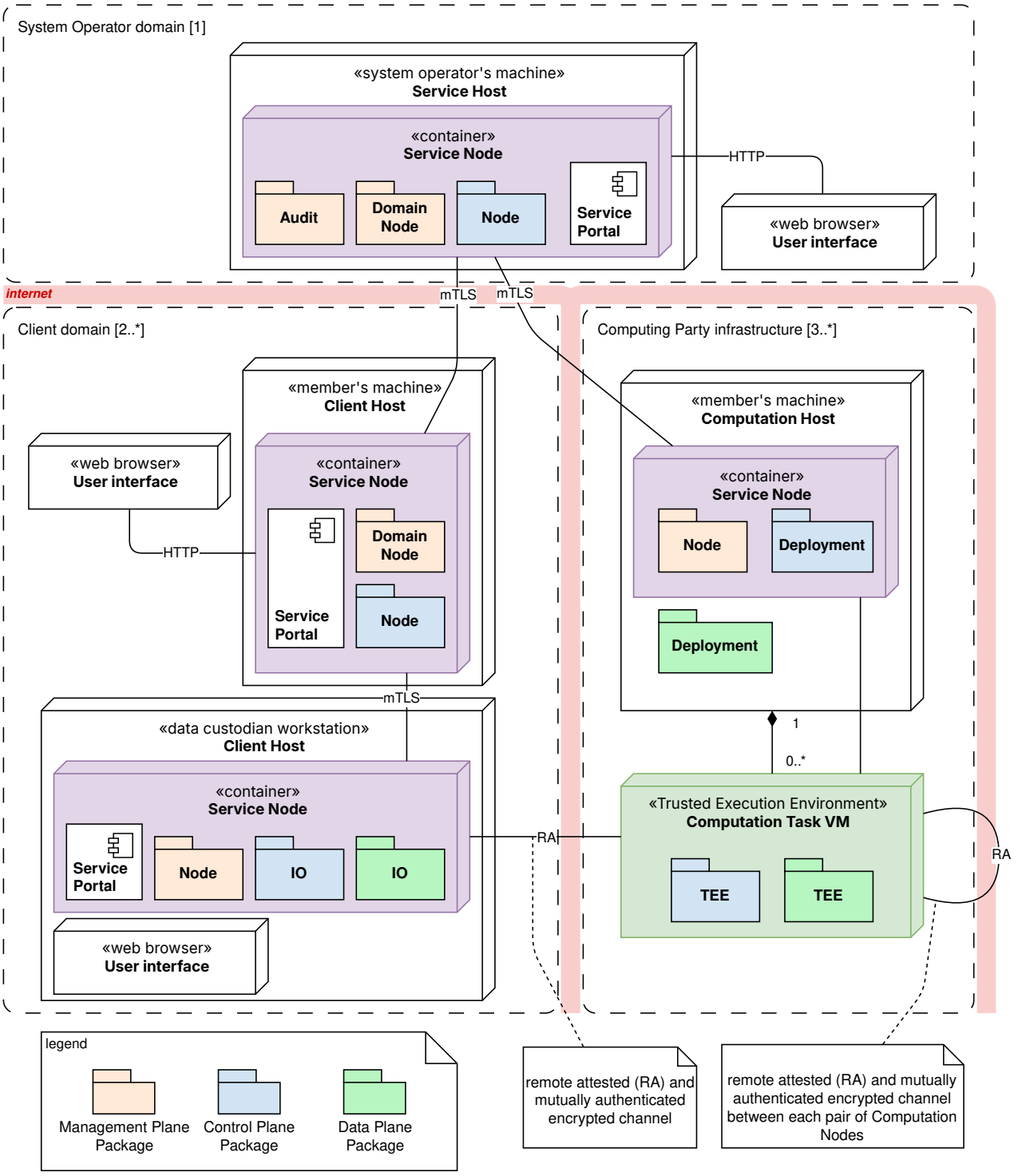


Figure 10. Example of a minimal deployment

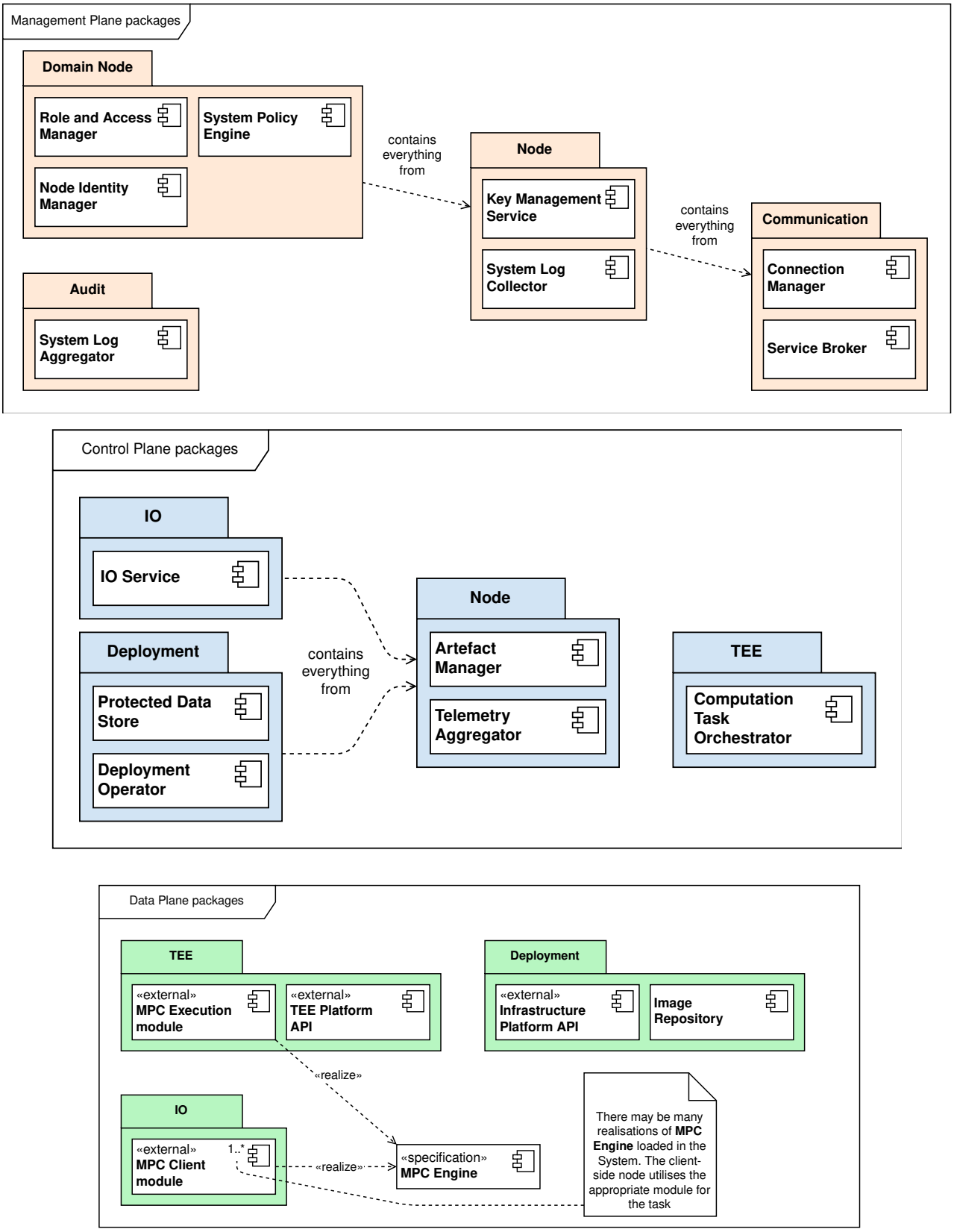


Figure 11. Component packages shown in Figure 10

Service Portal
Included in all Service Nodes which provide a web-based interface to its Users, enabling interactions with local and remote resources in a transparent manner. The **Computation Host** omits it in this example, indicating a headless deployment.

Management Plane Node package
Comprises core components that constitute a Service Node: communication facilities interaction with peer Service Nodes, identity and key management, and logging.

Management Plane Domain Node package
Contains all the components of the Management Plane Node package, with the addition of rich domain management capabilities, enabling fine-grained control over local policies, users, and roles (of peers and users).

Management Plane Audit package
Enables the extraction of logs from peers for auditing.

Control Plane Node package
Comprises core components for processing information associated with CTs.

Control Plane IO package
Supplements the previous package with capabilities for uploading or downloading data of CTs.

Control Plane Deployment package
Contains the capabilities of a CP's Service Node for controlling the deployment of CTs.

4.2.1.2 Computation Task Virtual Machine

The MPCaaS principle (PR-4) establishes that the CPs provide the Clients with a secure computation service. It implies that authorised entities can run tasks and interact with them. All such interactions are, however, strictly bound to a specific CT, which in fact should be totally isolated from one another. Hence, we encapsulate each CT at each CP within a dedicated VM based TEE (see Decision ADR-1 for additional details). The Service Node at the CP is responsible for managing only the VMs – i.e. their deployment with neccessary contents and teardown; in other aspects, the VM manages its own lifecycle internally.

Figure 10 illustrates the key elements of a System deployment.

Computation Task Virtual Machine (CT VM)
Positioned somewhere in the infrastructure which the CP operates. The exact placement depends on the infrastructure platform used by the CP[5].

Data Plane Deployment Package
Contains the platform-specific middleware which enables the Control Plane facilities to provision the VMs with the appropriate configuration.

Control Plane and Data Plane TEE Packages
Make up the trusted code of the CT VM. These include the Computation Task Orchestrator which enforces the CTA and lifecycle of the CT, the implementation of MPC Engine chosen for the task, and the middleware for interfacing with the TEE platform of choice to enable remote attestation.

[5] If the CP were to utilise a cloud service provider for their *Computing Party infrastructure*, it would use the appropriate infrastructure platform facilities to provision virtual machines on dedicated TEE-supported nodes. Otherwise, if the *Computation Host* were itself a TEE supported machine, it could simply deploy the CT VM within itself, e.g. by using QEMU.

Remote Attestation (RA) communication paths

Connect CT VMs with the outside world based on strict assumptions:

1. It only authorises ingress traffic from entities who provide proof of identity that matches with the CTA it holds;
2. If the connection is established to another CT VM (i.e. a counterpart CT VM instance), the source is required to prove the integrity of its code and CTA via remote attestation;
3. Similarly, for incoming connections, the CT VM offers proof of its integrity via remote attestation;
4. During the remote attestation procedure, the entities derive a key for an end-to-end encrypted and mutually authenticated channel.

The CP must make the CT VM accessible over a public IP and port for direct connection establishment, the details of which are also dependent on the infrastructure platform used. Communication in and out of the CT VMs preferrably involves minimal indirection that may inhibit latency or throughtput, as this is detrimental for the performance of MPC protocols.

Local Service Node and Computation Task VM communication path

are used by the CT VMs to engage in communication with its colocated parent Service Node to enable:

1. Acquisition of the CTA as the CT VM starts;
2. Provision of logs and telemetry;
3. Storage and fetching of Protected Data[6] .

The Computation Task VMs represent a deployment of a *VM image*. Images are files that contain the initial state of the VM, including all the enclosed software. In the System, images must be commonly identified and known to all entities. Anyone should be able to audit the code that makes up the image to assess that it behaves correctly. As the CT VM is remotely deployed, the remote attestation procedure proves to external entities that they are connected to a VM that is deployed based on the correct image.

The correct image is defined by the CTA (see Section 4.4). Images differ from one another w.r.t the Data Plane technologies employed by the task – most importantly the MPC Engine implementation which dictates the computational capabilities of the CT. There is at least one image per MPC Engine that is available to use, i.e. has been accepted and loaded in the process of augmenting the System. Images are distributed in their ready-to-deploy format in a static (partial) configuration, accessible for the infrastructure platform to instantiate. This is illustrated by Figure 12. All further configuration must be subsequently derivable from the CTA, which is loaded dynamically by the Service Node.

[6]The Protected Data at rest is stored outside the CT VM considering that data volumes might exceed the storage capacity of available managed TEE instances offered by cloud service providers. In this case the Protected Data is additionally encrypted with a TEE-based key (see Decision ADR-1).

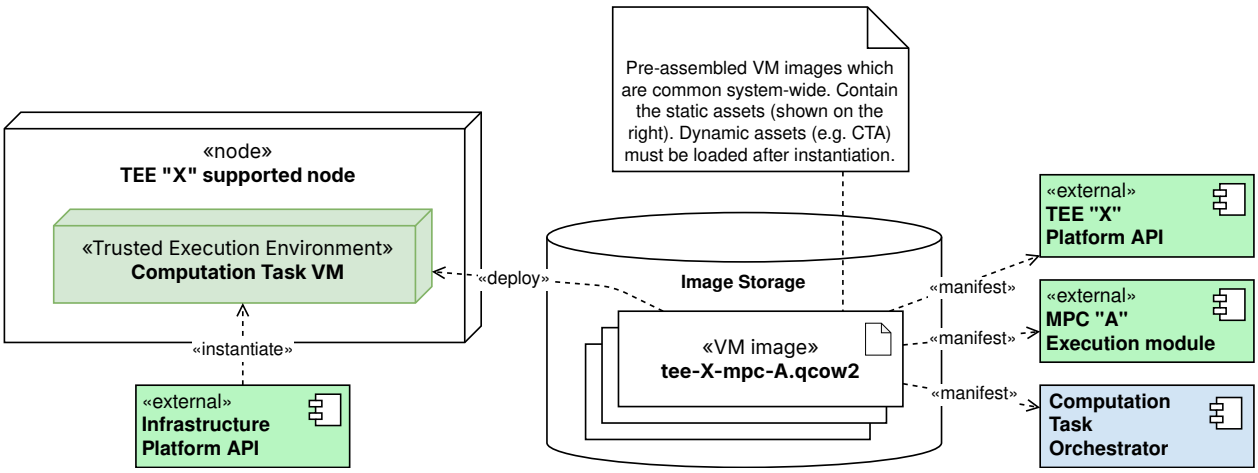


Figure 12. Deployment of a CT VM

4.3 Functional View

In this view we decompose the System into its core functional elements – components and interfaces – supplementing the deployment view (Section 4.2) and laying the foundations for the information view (Section 4.4). In this section we present a logical breakdown of the System, describing its responsibilities through the composition of its subcomponents and interfaces. By setting aside physical aspects we aim to present an abstract yet clear understanding of the System, facilitating design discussions, modularity analysis, and potential reuse.

> **Convention.** This section utilises the UML version 2.5 Component Diagram notation. Components (rectangles with the *component icon* in the upper right corner) represent independently deployable elements. Components may, however, be deployed in multiple instances, in different physical locations, and in part or in full, which is not made explicit in the diagrams. These aspects are explained in Section 4.2, allowing this section to focus on what is purely functional.

4.3.1 Top level functional elements

The System's top level functional elements are its subsystems, the responsibilities of which mirror the concept of Planes. The Service Portal component which serves as the gateway for user interactions is also one of the top level components. It is not regarded here as a subsystem component, as users depend on the facilities of multiple planes, hence the name *portal*, expressesing unified access to the System. The relationships between the top level functional elements are shown in Figure 13.

Management Plane «subsystem»
Responsible for the System level of:

- Provision of policy specification and enforcement capabilities;
- Local log collection and aggregation of remote logs for auditing purposes;
- Local key management;
- Node identity and role management;
- Internal user role management;
- Network connections and message exchange with Service Nodes.

Control Plane «subsystem»
Responsible for the CT's

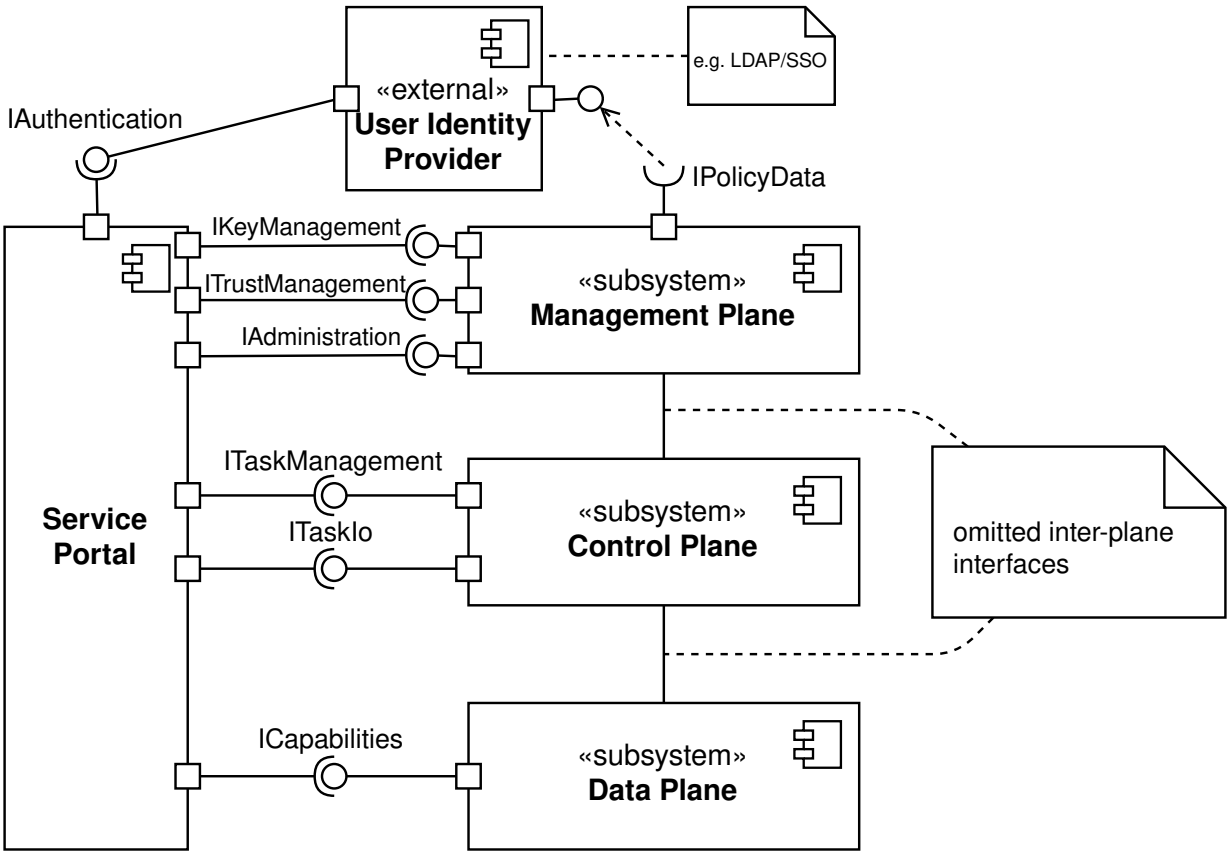


Figure 13. Top level view of the System components

- Lifecycle management;
- Consolidation process;
- Deployment;
- Orchestration;
- Input and output.

Data Plane «subsystem»
Responsible for the secure computing technology and platform integrations, i.e.

- Provision of CT VM;
- Enabling remote attestation;
- Provision of computation capabilities for the Control Plane «subsystem».

Service Portal
Responsible for the consolidation of intefaces necessary for user interactions and providing the user interface.

The User Identity Provider is marked as *external*, making it a dependency component. It represents a nonspecific authentication service for users of a domain that is likely to already exist within a prospective Member's infrastructure. Its integration is necessary to accommodate the distinction of organisation internal System roles (i.e. the mapping of physical persons to Signatory, Task Organiser, Data Custodian, or General Manager) in the case where the Member wants fine-grained control over who can do what. The Service Portal would redirect the user to the domain authentication service, obtaining an authentication token that enables System services to authorise subsequent actions.

4.3.2 Control Plane Subsystem

The functional decomposition of the Control Plane «subsystem» is shown in Figure 14. The following sections discuss the responsibilities of each component in depth.

4.3.2.1 Artefact Manager

The process of specifying the details of a CT boils down to the creation, distribution, access, and presentation of a variety of Computation Task Artefacts (CT Artefacts). The Artefact Manager is the functional element responsible for these operations. The Artefact Manager ensures that new artefacts adhere to System policy, and after consolidation of a CT is itself the source of CT policy. Artefacts comprise different data elements, e.g. (from Section 4.4):

- CTSC, CTSA;
- Digital signature;
- Control message, e.g. *intent to deploy* or *abort* a CT;
- PAK.

The decentralised governance principle (PR-2) and no SPoT principle (PR-3) impose several functionally significant considerations in terms of the system-wide processing of artefacts. To have decentralised governance over artefact admission implies that each involved participant has the autonomy to decide whether or not it should be processed. Each could have internally established policies that affect their decision. For this reason, it must be considered where the artefact is located and how it is processed depending on the location.

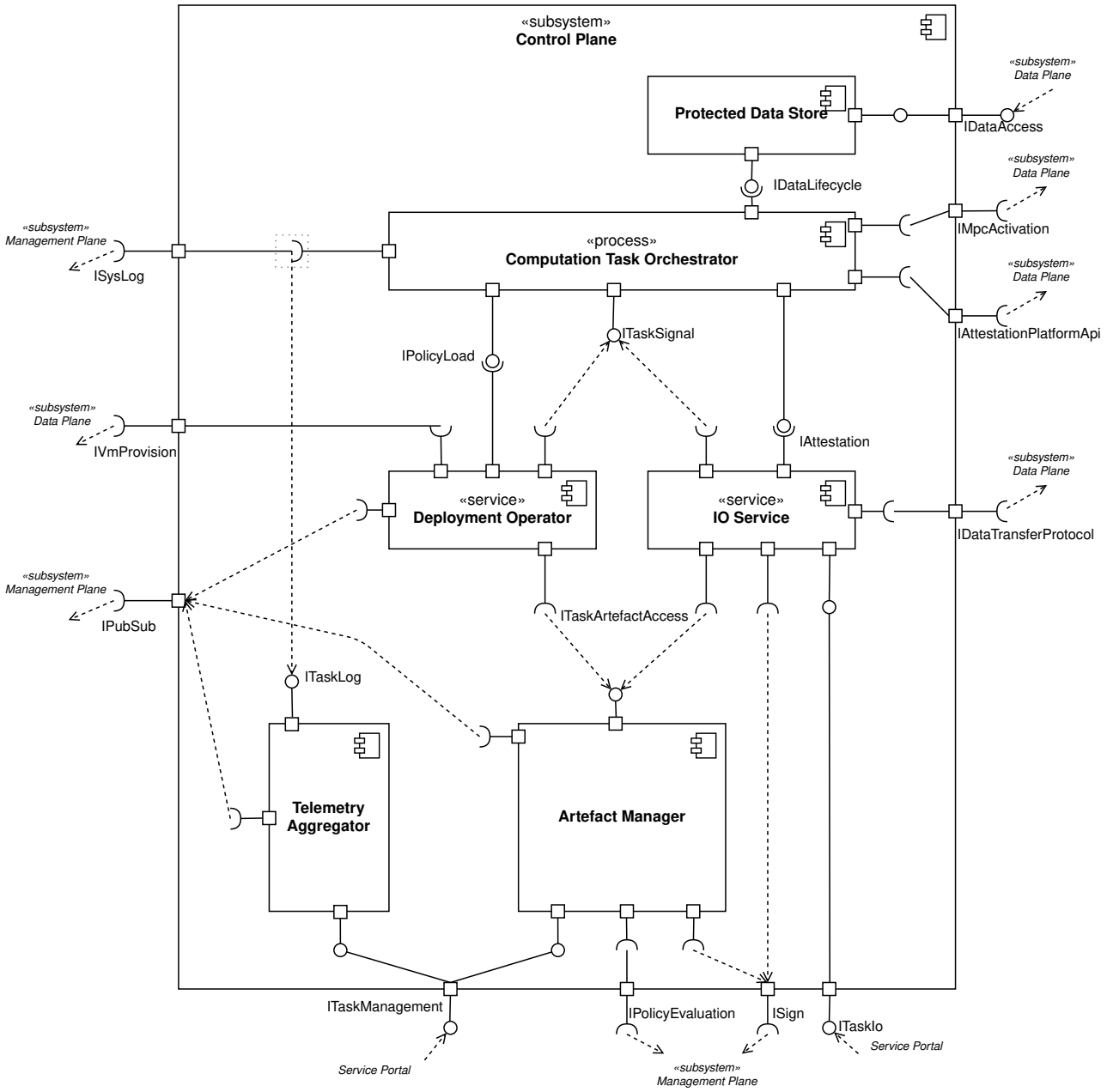


Figure 14. Control Plane «subsystem»

An artefact can be *created* at the Service Node or *received*, i.e. sourced externally. The exchange is facilitated by the Management Plane «subsystem» which is responsible for the delivery of messages (including those of artefacts) within the System. This enables decoupling the Control Plane «subsystem» from the Service Node internetworking and delivery mechanism.

The Artefact Manager has the following responsibilities.

1. Storage of artefacts.
2. Servicing requests for accessing or creation of artefacts. This involves creation of new task artefacts and presenting views of existing ones through the Task Management (ITaskManagement) interface. As the consolidation is finished, artefacts are made accessible to other components through the Task Artefact Access (ICtArtefactAccess) interface, i.e. to reference specific details.
3. Processing of artefacts as they are admitted, including validation of the adherence of artefacts to system policies or creating signatures.
4. Propagation of artefacts in the System by publishing them for other participants to process.

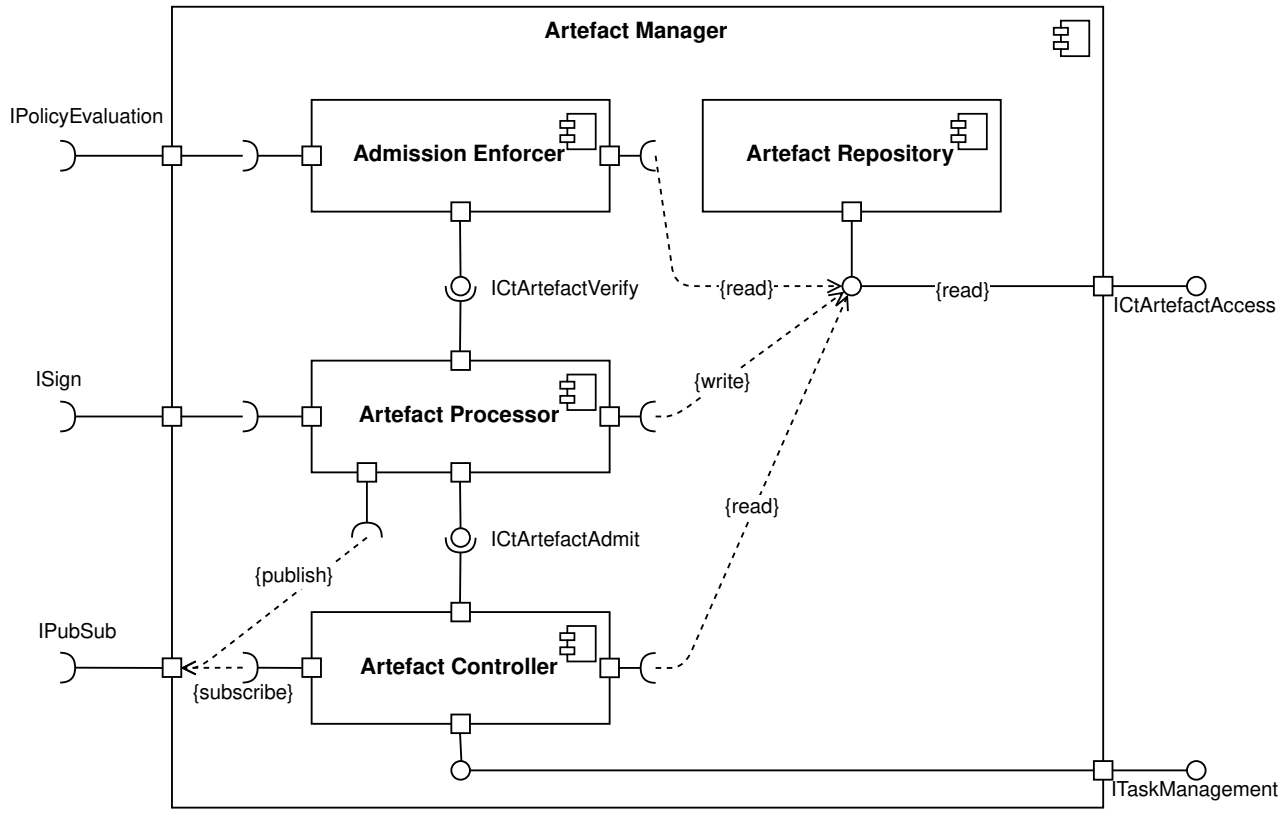


Figure 15. Decomposition of the Artefact Manager

Figure 15 Shows the functional decomposition of the Artefact Manager component.

Artefact Controller

Services requests from local users and handles incoming messages.

Responsibilities

- Authorisation of external access;
- Preparation of views, i.e. listings of CTs or detail views collating multiple artefacts;
- Handling artefact creation requests;

- Reception of artefacts from the message bus;
- Submission of artefacts for local processing and admission;

Interfaces – inbound

ITaskManagement
Provides artefact management capability of the user-facing task management interface.

Interfaces – outbound

Message Bus (IPubSub)
Used to receive any published messages of artefacts meant to be processed locally.

Task Artefact Admission (ICtArtefactAdmit)
Used to delegate artefact processing in the local Service Node

ICtArtefactAccess
Used to read artefacts for constructing views.

Artefact Processor

Handles operations regarding the admission and propagation of artefacts.

Responsibilities

- Validation of artefacts;
- Generation and reporting of errors and notifications;
- Obtaining signatures for artefacts;
- Formalisation of created artefacts, i.e. encoding of all task policies in their machine readable representation;
- Persisting of formalised artefacts, creation and and assignment of object identifiers if applicable;
- Determination of subsequent artefact recipients and publishing artefacts if applicable.

Interfaces – inbound

ICtArtefactAdmit
Accepts artefacts (received) or creation requests in well-defined formats that must subsequently be validated, formalised, and persisted.

Interfaces – outbound

Node Signature (ISign)
Used to generate signatures on formalised artefacts as needed to prove source and integrity.

IPubSub
Used to propagate artefacts to other entities. Artefact Processor Labels messages with the destined recipients to facilitate delivery.

Task Artefact Verification (ICtArtefactVerify)
Used for the delegation of artefact processing decisions.

ICtArtefactAccess
Used for writing or updating the persisted artefacts with newly processed information.

Admission Enforcer

Provides flexible verification of artefacts, supporting admission decisions.

Responsibilities

- Translation of artefacts in processing to relevant system policy queries, such as source authorisation of the artefact, e.g. if system policy permits artefacts to be created only by certain users or roles, or sourced from specific Service Nodes;

- Permission or prohibition of further processing of artefacts;
- Consistency verification, e.g. if an artefact violates business invariants of existing formalised artefacts.

Interfaces – inbound

ICtArtefactVerify
Accepts artefact data, returns a decision along with the reasons in the case of a rejection.

Interfaces – outbound

Policy Evaluation (IPolicyEvaluation)
Used for querying the Management Plane «subsystem» for policy decisions based on up-to-date system policies.

ICtArtefactAccess
Used for cross-referencing facts with existing artefacts.

Artefact Repository
Handles the local storage of artefacts.

Responsibilities

- Maintenance of a consistent state of artefacts;
- Management of relations and mappings of domain objects;
- Provision of necessary querying capabilities for colocated components.

Interfaces – inbound

ICtArtefactAccess
Internally provides read and write access to Artefact Manager components; externally exposes artefacts for consumption by other Control Plane «subsystem» components.

4.3.2.2 Deployment Operator

A CT is a manifestation of task artefacts. The procedure in which an artefacts manifest a CT VM is termed *task deployment*. Deployment is a functionality of the Control Plane «subsystem»; it controls and automates the provisioning of resources just as *PaaS*[7] providers provision applications without manual operations. The component dedicated to this is the Deployment Operator, shown in Figure 16. It is responsible for:

- Keeping tabs on CT VMs running in the local infrastructure;
- Authorising operations – deployment and aborting – of CT VMs;
- Assembling artefacts to provision the correct type of CT VM and bootstrap it with the established policies.

Similar to the messaging pattern described for Artefact Manager (see Section 4.3.2.1) a command must be propagated in the System to be processed at the CP, validating it ahead of execution. In more specific terms, CTs are meant to run within the infrastructure of the CPs, isolated from direct influence of the task stakeholders (Clients). Deployment offers limited control for external entities, e.g. an IP, to be able to abort a task.

The Deployment Operator depends on locally available artefacts at the time it receives a command. This is because the Artefact Manager already asserts the correctness and completeness of received artefacts, ensuring, among other aspects, that a locally admitted artefact may be

[7] Platform as a Service

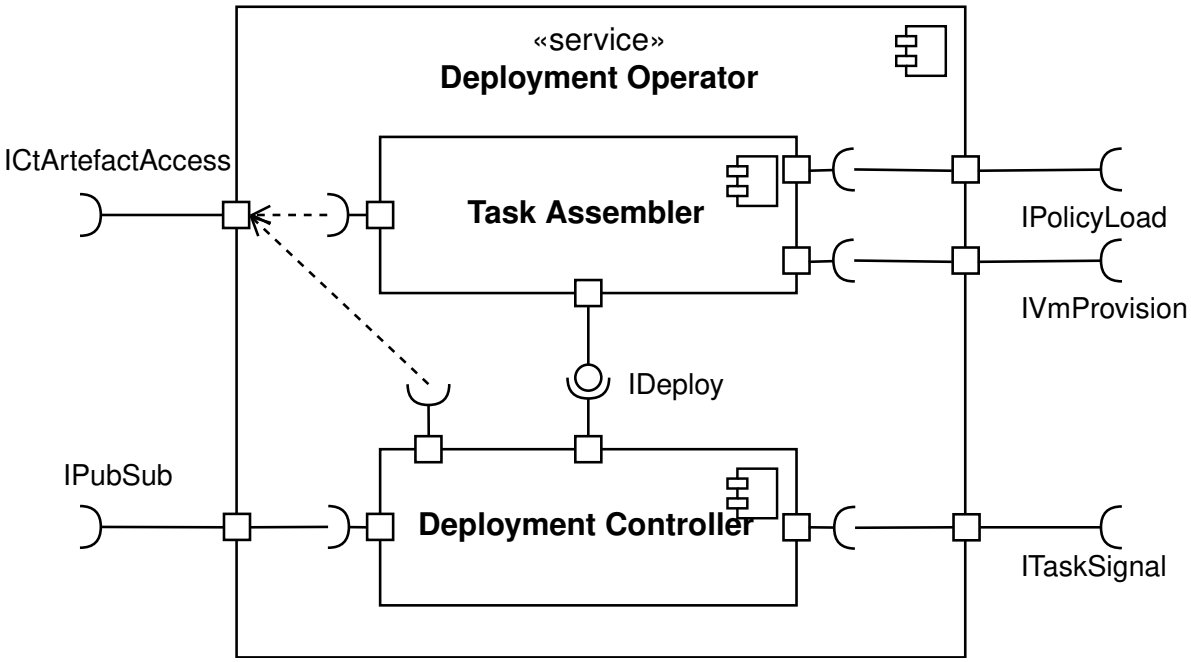


Figure 16. Decomposition of the Deployment Operator

used, relieving the Deployment Operator from further validation. This reflects the relationship between System-inherent policies and CT-based policies in *D1.1 Usage Scenarios and System Requirements, Section 2.4.3* – the Artefact Manager enforces the former when processing artefacts, making the artefacts (i.e. the latter) compliant and ready to implement.

Deployment Controller

Services commands associated to a deployment.

Responsibilities

- Authorisation of received commands based on artefacts;
- Invocation of the deployment procedure;
- Invocation of the abortion of the CTs.

Interfaces – outbound

Task Deployment Command (IDeploy)
Delegation of CT deployment.

IPubSub
Access to the message bus for receiving control messages and reporting errors.

Computation Task Signalling (ITaskSignal)
Used for signalling the colocated Computation Task Orchestrator, e.g. to initiate a graceful shut down in the case of an abort.

ICtArtefactAccess
Used as the source of truth for authorisation of commands.

Task Assembler

Carries out the deployment procedure.

Responsibilities

- Consulting artefacts to prepare the intended CT configuration;
- Providing the CT VM with the specified image containing the necessary instrumentation;

- Bootstrapping the Computation Task Orchestrator with the relevant configuration.

Interfaces – inbound

IDeploy
Enables the invocation of the deployment procedure, expecting an identifier to the subject artefact(s).

Interfaces – outbound

ICtArtefactAccess
Required for fetching details of the deployment configuration (e.g. image identifier) and dynamic configuration (i.e. task policy).

Virtual Machine Provisioning (IVmProvision)
Used for the operation of the platform-dependent API to dynamically provision CT VM instances.

Task Policy Loading (IPolicyLoad)
Used for the initial configuration of created CT VMs, embedding the policies in the Computation Task Orchestrator.

4.3.2.3 Computation Task Orchestrator

After deployment, the Computation Task Orchestrator autonomously carries out the functions of an individual CT. It should be noted that as the CT follows the MPC paradigm, any active CT comprises multiple Computation Task Orchestrator instances, one at each of the counterpart CPs of the CT. It is necessary for each of the instances to:

- Strictly follow its own view of the task policy;
- Make decisions consistently in unison with its counterparts;
- Make each decision autonomously, based on locally available information.

In *D1.1 Usage Scenarios and System Requirements, Section 4.4*, the abstraction used to describe a CT was a finite state machine that is derived from the CTS. The component responsible for the management of this distributed state machine is the Task State Machine, the relationships of which to the other components of the Computation Task Orchestrator are shown in Figure 17.

The decomposition presented in Figure 17 comprises the following functional elements.

Task State Machine
Manages the CT lifecycle – its state and state transitions.

Responsibilities

- Acting on internal and external signals. An internal signal might be a response to temporal cue (e.g. CT reaching its deadline) or to coordinate distributed state, an external signal might be an IO procedure or a call to abort the task;
- Authorisation of external signals w.r.t the state and policy of the CT;
- Delegation of the creation of necessary resources. These resources may be an instance of an MPC execution, handling the execution of an algorithm, a communication path between processes, or preallocated data elements in storage;
- Generation and publishing of logs of the CT progress.

Interfaces – inbound

ITaskSignal
Allows the Task State Machine to process external signals that advance its state.

Interfaces – outbound

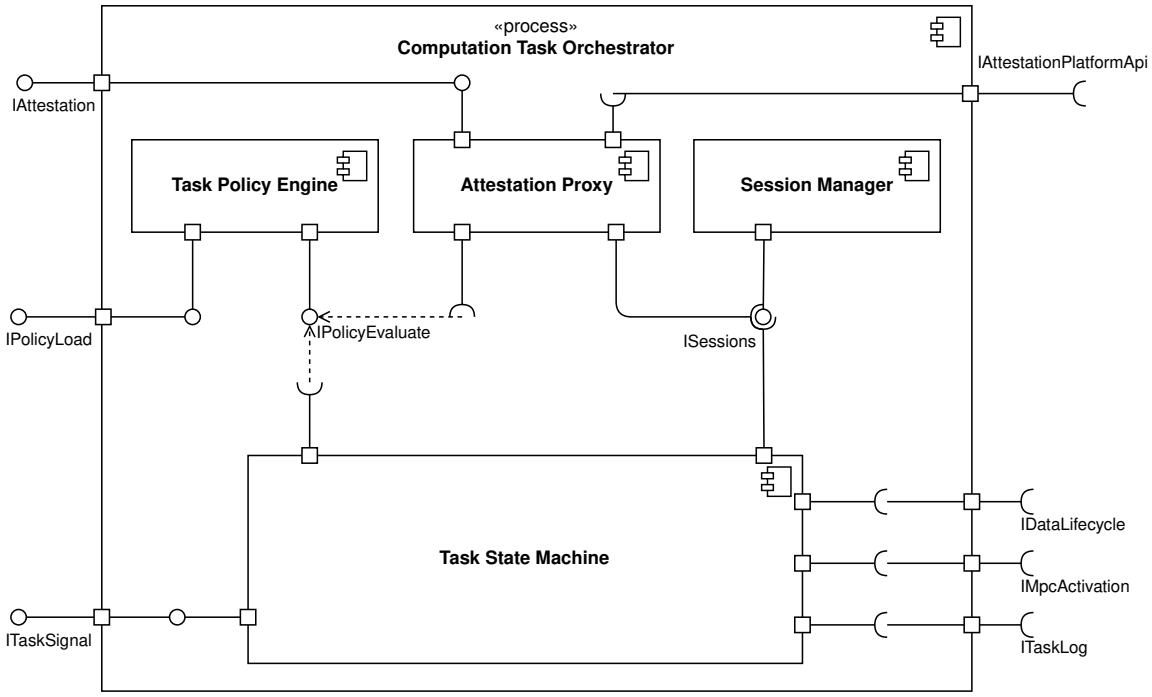


Figure 17. Decomposition of the Computation Task Orchestrator. The «process» stereotype implies that the component is transactional. As such, an instance of it has its own state, activity period, business logic

IPolicyLoad
: Used for the derivation of the structure and rules of the state machine. For example, if the CTS defines n Input Data elements to be uploaded, then the Task State Machine will not advance to begin execution until this condition is met.

Session create-read-update-delete (ISessions)
: Used for the association of sessions with communication paths, i.e. amend attributes of established sessions with rules that permit or route communication between a remote entity and a local MPC process.

Data Lifecycle (IDataLifecycle)
: Used for the management of the lifecycle of the Data Slots (Slots) (see Section 4.3.2.5) specified by the Task Policy Engine.

MPC Activation (IMpcActivation)
: Used for spawning new instances of MPC processes.

Task Logging (ITaskLog)
: Used for supplying both telemetry and system logs.

Session Manager

Manages the access of established connections to communicate with task-internal processes.

Responsibilities

- Maintenance of a session storage for external connections, e.g. a client or peer Computation Task Orchestrator;
- Enabling linking *communication paths* with sessions. For instance, as an MPC process is started in different Computation Task Orchestrator instances, they require direct communication with each other. The peers have already established a trust via the Attestation Proxy and hence have an active session. A session should be labeled with the necessary information to enable routing of communication to and from one socket

in one instance and the counterpart socket in another instance. This way, the Control Plane «subsystem» is in control of the communication between Data Plane «subsystem» components.

Interfaces – inbound

ISessions

Provides functions to read session metadata and create or modify sessions.

Task Policy Engine

Holds an immutable view of task policies.

Responsibilities

- Serving as an authoritative source of policy (as both the *Policy Decision Point* and *Policy Information Point*) within the deployed Computation Task Orchestrator;
- Provision of means for external entities to verify the running policy;
- Provision of means for other components to evaluate and enforce the running policy.

Interfaces – inbound

IPolicyLoad

Enables initial, *write-once* configuration of policy by a parent Deployment Operator. Subsequently enables policy queries.

IPolicyEvaluation

Enables evaluation of policies.

Attestation Proxy

Provides remote attestation capability and secure channels to external connections.

Responsibilities

- Enabling remote connections to verify the integrity of the Computation Task Orchestrator and its host CT VM;
- Establishment of mutually authenticated channels and sessions;
- Offering first layer defence w.r.t external connections, limiting the permitted connections to those identities, which are permitted by policy to interact with the Computation Task Orchestrator.

Interfaces – inbound

Remote Attestation (IAttestation)

Provides the TEE platorm agnostic functions to verify integrity of the secure environment and form mutually authenticated encrypted channels.

Interfaces – outbound

IPolicyLoad

Used for obtaining the cryptographic identities of permitted connections.

TEE platform API (ITeePlatformApi)

Used for invoking the TEE platform specific functions to enable environment integrity verification via the IAttestation interface.

ISessions

Used for the creation of new sessions, persisting the authentication information post attestation for subsequent communication.

4.3.2.4 IO Service

The System procedurally handles the input and output of Protected Data through the Control Plane «subsystem» rather than the Data Plane «subsystem» due to the aspect of machine enforcement necessary for these actions (requirement BUS-4.2). Based on the division of responsi-

bilities outlined in Section 4.3.1, only the former is capable of authorising external interactions with the CT, e.g. restricting access to certain entities to extract Output Data. This is further supported by the requirement SYS-10.2 stating that such actions must be logged.

The IO Service is, however, dependent on the Data Plane «subsystem». More specifically, the transformation of data into the protected representation required by the MPC Engine is delegated to respective MPC Engine realisation. With consideration to the modularity requirements of the Data Plane and contained technologies (SYS-7.4 and BUS-5.4), this is one of the important *integration interfaces* that enables the data plane modularity principle (PR-6).

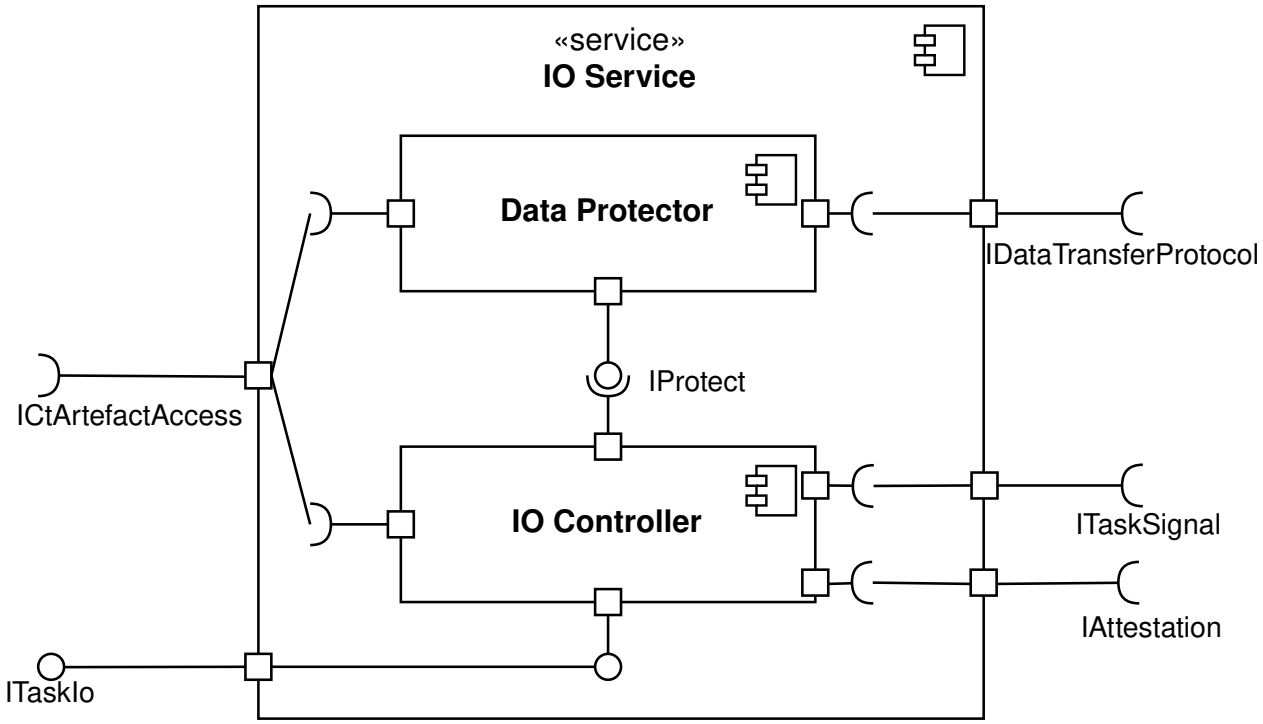


Figure 18. Decomposition of IO Service

Figure 18 shows the decomposition of the IO Service component.

IO Controller

Services requests to transfer protected data to and from the CT.

Responsibilities

- Using task artefacts to authorise the caller and determine the target nodes;
- Running remote attestation, establishing a secure connection with the target TEEs;
- Signaling the orchestrator to prepare for the transfer and obtains a communication channel to each of the MPC processes;
- Delegation of the Data Protector for the protection and transfer of data.

Interfaces – inbound

Task Input Output (ITaskIo)

External interface for local users to invoke a Protected Input/Output Data transfer procedure. It expects the caller to specify the task artefacts for the CT and the target data specification in question. The interface further accepts the plaintext data to be provided for input transfer; for output transfer procedures, the caller is returned

the plaintext data. This interface must authorise the caller w.r.t to the task artefacts, validating that the caller is permitted to operate on the given CT and data.

Interfaces – outbound

ICtArtefactAccess
Used for reading the data specification (for authorisation) and the task specification (for resolving Computing Nodes and acquiring the remote attestation measurement).

IAttestation
Used for running the remote attestation procedure and subsequently establishing a direct connection with the TEEs running the CT.

ITaskSignal
Used for notifying the MPC orchestration layer of the intent to proceed with a data transfer procedure, acquiring a socket to an MPC runtime.

Data Protection (IProtect)
Delegates the conversion of data from plaintext to MPC Engine specific representations or vice versa.

Data Protector
Transforms data from plaintext to MPC protocol messages and back.

Responsibilities

- Using task artefacts to determine the appropriate MPC protocol scheme;
- Using task artefacts to determine the format and layout of the plaintext data;
- Parsing the plaintext data w.r.t its format and layout;
- Binding the MPC Engine specific implementation to upload or download the protected representation of data.

Interfaces – inbound

IProtect
Accepts (1) sockets to exchange messages with the MPC runtimes; (2 – *for input procedures*) plaintext data in a well-defined structured data format, (2 – *for output procedures*) returns plaintext data; (3) an ID to the data and task specification.

Interfaces – outbound

ICtArtefactAccess
Used for reading (1) the task specification for determining the MPC Engine to use for communication, and (2) the data specification for the expected layout of the data.

Data Transfer Protocol (IDataTransferProtocol)
Calls the appropriate functions provided by the MPC Engine for converting individual values between plaintext data types and their protected counterpart representations.

4.3.2.5 Protected Data Store

The distinct nature in which Protected Data associated with a CT must be handled warrants specialised functional elements. Based on the requirements we know that:

- Input Data and Output Data is always well-defined in the CTS w.r.t expected structure, its supplier or receipient, the Algorithm it is processed or produced by.
- Any access to either must be authorised based on these rules, implying that all such instances of data must be discrete – subject to individual control.

- Data is provided in no specific order or time, being dependent on the users initiating an IO procedure, hence it must be stored ahead of processing.
- The discrete data instances must also adhere to a data lifecycle depending on their classification, e.g. Input Data must be deleted upon the completion of the algorithm execution it is processed by.
- Protected Data is an MPC-specific representation of data. We assume a case of data distribution where each logical data element is split into multiple physical elements (e.g. *secret shares*), each held by one CP.

The Computation Task Orchestrator can identify the discrete data elements it expects to receive or produce up front. We say that each such element corresponds to a Slot. A Data Slot functionally enables the writing and reading of arbitrary data to and from controlled regions of local storage. The Computation Task Orchestrator controls the lifecycle and access of these Slots. To exemplify:

- As the Computation Task Orchestrator is created, it allocates a Slot S for an Input Data table T expected from Data Custodian DC.
- S is associated with a data lifecycle and access policy, and state which is initially '`empty`'.
- The Computation Task Orchestrator, upon receiving a signal (over ITaskSignal) from DC to start sending T:
  - Asserts that DC is the intended provider of the respective S, otherwise rejects the request;
  - Asserts that S is `empty`;
  - Locks S by switching it to an `in use` state;
  - Gives the MPC Engine exclusive write access to S (over IMpcActivation) for the upload procedure;
  - If the procedure succeeds (in all counterpart CT VMs), S is marked as `ready`, otherwise S is rolled back to `empty`.

By invoking an algorithm execution, the IMpcActivation interface is supplied with the respective sockets through which the MPC Engine can load data. This solution is in conformance with the plane separation principle (PR-1) and technology integration strategy principle (PR-5), as it lowers the barrier for integrators of MPC technologies while achieving the required control measures.

Protected Data Store (shown in Figure 14) is responsible for the storage, access, and destruction of Slot contents owned by a Computation Task Orchestrator. It provides the IDataLifecycle interface for Slot initialisation and destruction. It provides the Data Access (IDataAccess) socket interface for reading or writing arbitrary binary data of a Slot.

4.3.2.6 Telemetry Aggregator

It is required[8] for select individuals to be kept up to date with the progress and status of the Computation Task Orchestrator to facilitate user interactions and observability. After deployment, however, the Computation Task Orchestrators are remote resources, requiring a solution for monitoring. This is different from the visibility offered by the Artefact Manager, in that CT Artefacts represent prescriptive metadata, whereas monitoring is purely descriptive.

This is enabled by Telemetry Aggregator, shown in Figure 14. Telemetry Aggregator is responsible for the collection, distribution, and presentation of operational telemetry, i.e. CT runtime events. Specifically:

[8] BUS-5.2 and SYS-6.3

- Computation Task Orchestrators provide internal events to the colocated Telemetry Aggregator over the ITaskLog interface.
- Telemetry Aggregator processes the events, e.g. filtering, enriching with auxiliary information, forming messages.
- Through the IPubSub interface, the telemetry data is made available to peers.
- The Telemetry Aggregator collects and stores received messages.
- Through the ITaskManagement interface, the component enables users to track progress of Computation Task Orchestrators.

Telemetry plays a significant role in the coordination of the actions of the Data Custodians and provides transparency to the Task Organisers. Coordination involves the System indicating when it expects manual operations, most notably IO. As Input Data upload involves the validation of data quality (see *D1.1 Usage Scenarios and System Requirements, Section 4.4.2*), the Data Custodian would be made aware of the verdict through telemetry. The transparency aspect enables the Task Organisers to assert whether the Computation Task Orchestrator is progressing in schedule to meet the deadlines, get an overview of any blockers, and be informed of any errors.

4.3.3 Data Plane Subsystem

The Data Plane «subsystem» comprises several dynamic components which augment the System with secure computation capabilities (differing in qualities, e.g. functionality, security model, or performance) or provide means for the System to call low-level platform or infrastructure APIs. According to the data plane modularity principle (PR-6), the manifestations of these components depend on the System deployment configuration. Hence, these components represent the specification of modular components of the Data Plane «subsystem».

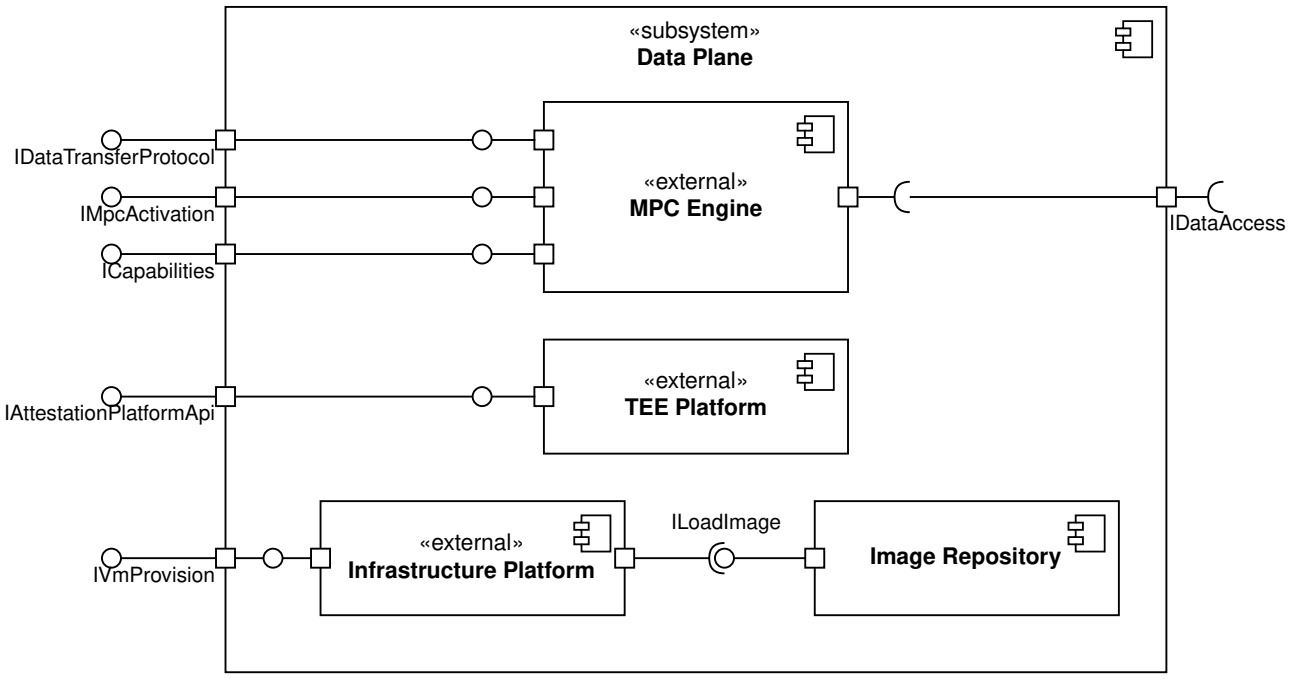


Figure 19. Data Plane «subsystem»

Each of the three «external» components shown in Figure 19 are abstractions denoting existing technologies or services. We describe the behavioral semantics of these components without

making assumptions about inner structure. Further, we provide high level descriptions of their interfaces, revealing integration requirements of interest to technology vendors and integrators.

4.3.3.1 MPC Engine

The System incorporates its suite of MPC functionality through one or several MPC Engine implementations. An MPC Engine is based on an *MPC framework*[9] (*see [2, p. 41]*) which adheres to the following criteria.

Human-readable domain specific language (DSL). The framework is accompanied by a compiler or interpreter that translates a length of human-readable code into MPC-based data analysis programs [1]. The DSL in which the algorithms are defined should enable the transfer of only the self-contained and meaningful part of the user-specified data analysis. High-level languages can facilitate this by embedding commonly-used features into standard libraries which are distributed along with the framework. Besides general purpose compilers, purpose-specific MPC protocols (e.g. private-set intersection) could be integrated as (or into) an MPC Engine, provided its usage can be represented with directives parametrised with input and output sources.

Stateless self-contained execution. The System regards an execution of an MPC program as a transaction. The framework should provide a simple execution interface that readily executes any program in its DSL as a unit of work. Any resources the execution should need are provided up-front. An execution should not depend on external services, and multiple executions should be independent from one another.

Outsourcing-based MPC. The framework must facilitate the transfer and conversion of Protected Data between the *server* and the *client*. This has several implications. First, the framework should distinguish between protocol-specific private data types and the counterpart public data types (e.g. integers, floats) that it supports. Software utilities should be provided for conversion between these types, enabling clients in possession of public data to provide it to servers in a protocol-specific representation and vice versa. As some MPC protocols require this procedure to involve several rounds of communication, the System facilitates this by regarding the uploading and downloading of data as an MPC program running between the client and the server. In this special execution, the framework needs to be able to read and write the private data types to and from its runtime context and at-rest.

The MPC Engine is a functional element describing the mandatory capabilities of any one MPC module that is loaded into the System. An individual implementation of an MPC Engine must be uniquely identifiable System-wide, in terms of its *dynamic* capabilities, e.g.:

- Runtime environment prerequisites inherent to the utilised MPC protocol, e.g. number of parties and networking requirements that the Control Plane «subsystem» should fulfil;
- Supported DSL or directives;
- Supported data types.

[9] Note that the some components of the Control Plane «subsystem» describe functionalities inherent to several *MPC platforms* on the market. Different platforms present various solutions for storage (Protected Data Store), server- (Computation Task Orchestrator), and client-side (IO Service) facilities. However, none of the existing platforms are suitable as a drop-in component in their present form. Hence, the current version of the document does not attempt to describe an integration with a specific platform. Prototyping may show whether existing platforms (in part or in full) can be integrated as the components described here.

It should be expected that one such implementation constitutes one well-defined feature set that is usable as-is, without additional configuration necessary. For example, if a framework supports numerous protocols (for different amounts of parties, different language features, data types, etc.) then each of these should be considered as a separate MPC Engine.

Responsibilities

- Augmentation of the System with MPC functionality, including the MPC execution environment with protocols, the DSL, the data protection mechanism;
- Provision of hooks for IO Service to enable the transfer of Protected Data.

Interfaces – inbound

IDataTransferProtocol
Provides the IO Service with functions to transform public data types to protected data types, in conjunction with the communication protocol necessary to transfer the Protected Data to the remote MPC Engine instance.

IMpcActivation
Dictates how execution is invoked and parametrised with communication channels to counterpart MPC Engine instances or remote IO Services, and local Protected Data access.

Capability Query (ICapabilities)
Enables querying specific properties of the MPC Engine (e.g. supported data types) for validation of relevant user inputs. For example, as a Data Custodian selects a set of data to upload, the IO Service can locally assert whether the input is well-formed. If necessary, this interface can be purposed to validate the MPC Engine specific details of the CT Artefacts during the CTS initiation, e.g. to verify that an Algorithm is syntactically correct, or that a CTS contains the correct amount of CPs.

Interfaces – outbound

IDataAccess
Required for runtime access to Protected Data.

4.3.3.2 TEE Platform

The TEE Platform offers the capability of TEE-vendor specific remote attestation. This component is inherent to the VM-based TEE that hosts the CT VM. Contrary to MPC Engines, which should be designed around the interface contracts defined by the System, the various TEE Platforms are integrated by the System. Due to the variations between the currently available TEE technologies, it is more reasonable to develop different realisations of dependent System components (e.g. Attestation Proxy) that depend on it.

The ITeePlatformApi interface shown in Figure 19 is determined by the specific TEE vendor. The interface must enable the colocated application to generate a measurement of the running VM.

4.3.3.3 Infrastructure Platform

The Infrastructure Platform (see Figure 19)) represents an abstract component that enables programmatic manipulation of the infrastructure, e.g. provision of VM resources and network configration. Such capabilities are often offered by cloud service providers.

The IVmProvision combines the creation of a VM and configuration of a firewall to expose the VM to external connections. The component should be able to source base images for the created

VMs over the Virtual Machine Image Retrieval (ILoadImage) interface.

4.3.3.4 Image Repository

The Image Repository is a simple storage facility for *pre-baked* VM images which contain the Control Plane «subsystem» instrumentation along with external components, including the MPC Engine implementation. It is responsible for maintaining a registry of the available VM images and providing access to them over the ILoadImage interface.

4.3.4 Management Plane Subsystem

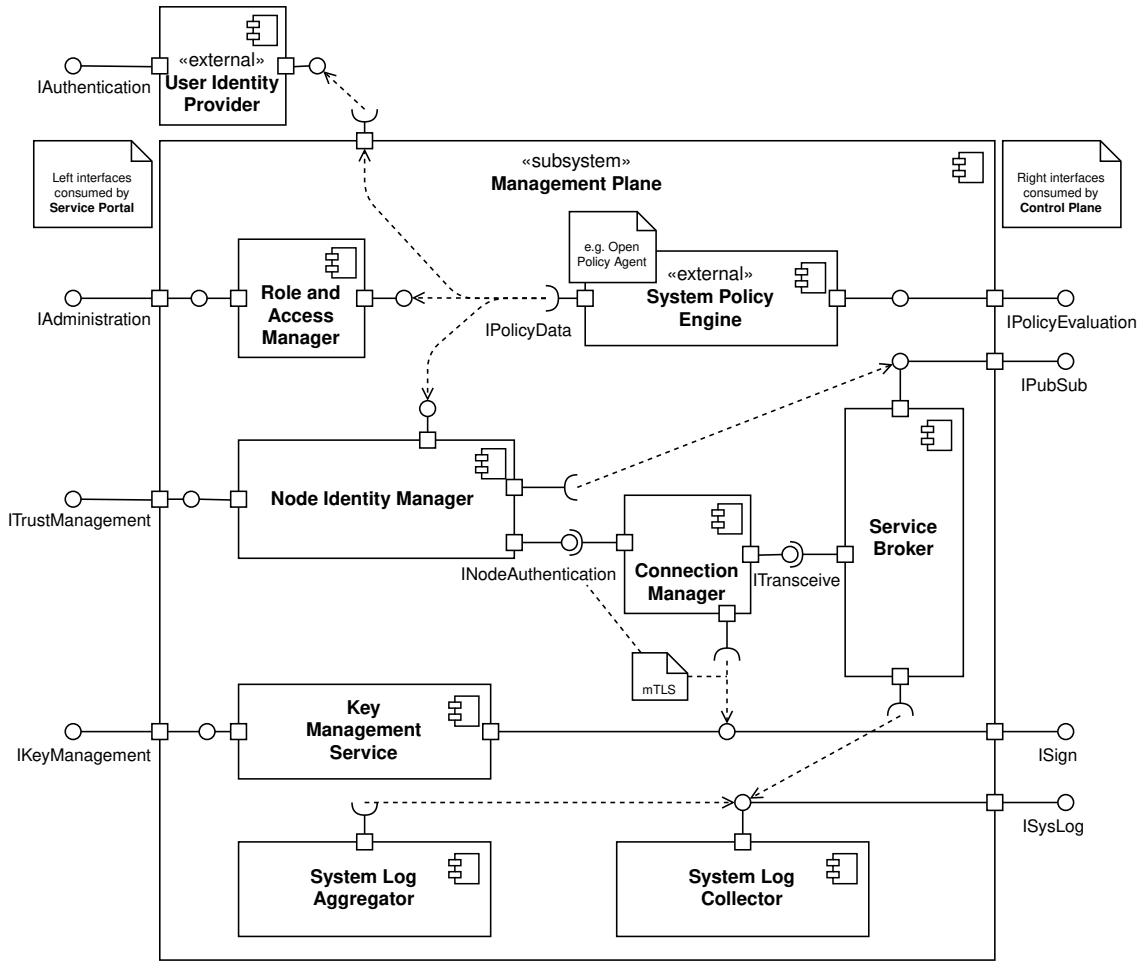


Figure 20. Decomposition of the Management Plane «subsystem».

Here we describe the tentative structure of the Management Plane «subsystem». Figure 20 outlines its inner components.

4.3.4.1 System Policy Engine

Responsibilities

- Provision of a unified way to specify system policy rules;
- Serving as the policy decision point (PDP)[10];

[10]Policy decision points (PDP), policy information points (PIP) are, among others, standard components of an

- Integration with data sources (policy information points (PIP)) to enable rich policy definitions.

Interfaces – inbound

IPolicyEvaluation
Enables the evaluation of terms against system policy.

Interfaces – outbound

Policy Information Access (IPolicyData)
Integrates the PDP with auxiliary real-time information from different PIPs.

4.3.4.2 Role and Access Manager

Responsibilities

- Storage of the mapping of users to system roles;
- Storage the mapping of child Service Nodes to roles;
- Permitting authorised users to modify role assignments on the fly.

Interfaces – inbound

Administration (IAdministration)
Enables the management of both user and Service Node roles and access permissions.

IPolicyData
Enables querying roles and access permissions for policy decisions.

4.3.4.3 Node Identity Manager

Responsibilities

- Storage of (local view of) identities of neighbouring Service Nodes and remote Service Nodes;
- Provision of certificate based operations, e.g. authentication of peers (prerequisite to mTLS);
- Enabling admission and comparison of identities which have been acquired by out-of-band means.

Interfaces – inbound

Trusted Certificate Management (ITrustManagement)
Used for manipulating the identity store, either for adding new trusted certificates, rotating certificates, or for specifying the certificates of neighbouring Service Nodes.

IPolicyData
Enables querying the identity information of nodes for policy decisions.

Service Node Authentication (INodeAuthentication)
Enables components to verify identity claims of Service Nodes.

Interfaces – outbound

IPubSub
Used for propagating the local view on trusted identities to other Service Nodes.

attribute-based access control (ABAC) model [4].

4.3.4.4 Connection Manager

Responsibilities

- Physical networking in the Service Node network;
- Maintenance of a view of the IP addresses of its neighbouring nodes.

Interfaces – inbound

Message Transmit and Receive (ITransceive)
Enables network-transparent data exchange, allowing messages to be addressed based on identities.

Interfaces – outbound

INodeAuthentication
Used for the authentication of a neighbouring Service Node for establishing a secure mutually authenticated channel.

ISign
Used for providing the identity of the local Service Node for establishing a secure mutually authenticated channel.

4.3.4.5 Service Broker

Responsibilities

- Coordination of inter-node message exchange;
- Routing messages to successive nodes;
- Relaying incoming messages, based on topic, to relevant local components;
- Logging seen messages.

Interfaces – inbound

IPubSub
Enables System components to publish messages and subscribe to messages on various topics which are transported over the Service Node network.

Interfaces – outbound

INodeAuthentication
Used for the authentication a neighbouring Service Node for establishing a secure mutually authenticated channel.

ISign
Used for providing the identity of the local Service Node for establishing a secure mutually authenticated channel.

4.3.4.6 Key Management Service

Responsibilities

- Secure storage and management of private keys;
- Enabling the utilisation of private keys for the digital signing of arbitrary data by other local components;
- *(possibly)* Integration with an online Hardware Security Module (HSM).

Interfaces – inbound

Local Cryptographic Key Management (IKeyManagement)
: Enables the creation and rotation of cryptographic keys.

ISign
: Enables using specific keys for signing.

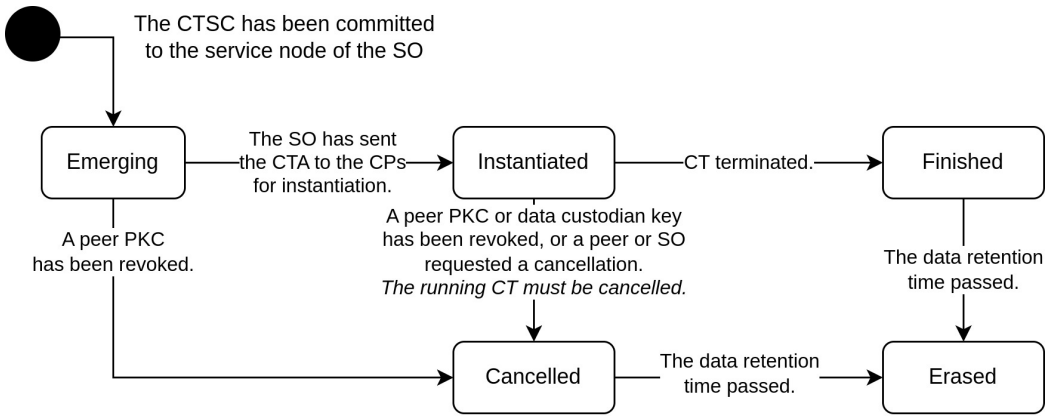


Figure 21. The lifecycle of a CTA begins when a Task Organiser submits a CTSC to the System through their Service Node

4.4 Information View

4.4.1 Computation Task Agreement (CTA)

The CTA and its components are the central data elements of the JOCONDE System. The CTA assembles all information which is required to instantiate and interact with a CT. The client-facing functionality primarily focuses on the creation of the CTA and the interaction with the resulting CT. The requirements for the CTA are described in *D1.1 Usage Scenarios and System Requirements, Section 4.4.1*.

4.4.1.1 Lifecycle

The lifecycle stages of the CTA are as follows (Figure 21):

Emerging

Asynchronous construction phase, visualised in Figures 22 and 34.

CTSC Creation

The main Task Organiser of a CTA first creates the CTSC within their Service Node, compiling all the information as shown in Figure 23. This information is negotiated by all Peers out-of-band. When the CTSC is finished, it will be propagated up to the Root Service Node. There, a newly-generated UUID will be assigned and returned to the initiating Service Node. The whole chain of Service Nodes now stores the CTSC and its UUID. From this point on, the Peers can continue with the creation of the CTSAs, and the SO can start verifying the emerging CTA. The Task Organiser maybe needs to forward the UUID to their team depending on the configuration (Section 4.4.1.5), i.e. the Signatories, Data Custodians, and Data Analysts.

CTSA & Node Signature Creation

Each Task Organiser can now refine the Signatory, Data Custodian and Data Analyst roles by creating the CTSA, as per Figure 24. This information is not part of the CTSC itself: each Peer only needs to know what other *organisations* participate in the computation and what overall permissions they have, not the specific Signatory or Data Custodian identity and permission. Hence each Peer only signs their own refined identities and the company/department identity of the other Peers.

Upon completion, the addendum is sent to the Root Service Node, thereby possibly replacing earlier addenda from this Peer.

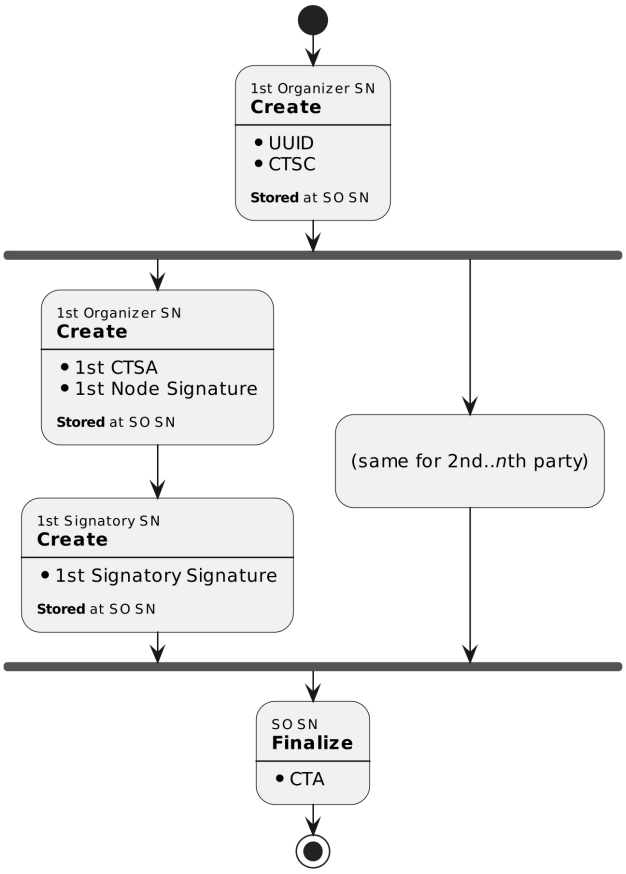


Figure 22. An *emerging* CTA is put together piece by piece. One Task Organiser creates the CTSC. Afterwards every Peer continues with the creation of the CTSA. In the end, the SO reviews the accumulated data and finalises the CTA if they accept the specification

Signatory Signature Creation

The signatory needs to review the CTSA on their Service Node and add their signature if it is correct, otherwise ask the respective Task Organiser to do corrections. The signature shall cover the hash of the CTSA. The application may provide to export the hash to a PDF file, which the Signatory can print, sign, scan and import into the Service Node. The implementation may also support electronically signing, e.g. creating an ASiC-E container or similar, directly within the user interface.

CTA Finalisation

The SO needs to manually approve the CTSC and CTSAs, or reject the addendum with appropriate instructions regarding how to fix the addendum. When all CTSAs are approved, the CTA is finalised and ready for instantiation. See Figure 25.

Instantiated

The Root Service Node forwards the CTA to the Service Nodes of the CPs. They will automatically prepare the CT VMs and make them accessible via the Internet to the Data Custodians and Data Analysts (Section 4.4.6). An instantiated CT VM itself goes through the following states. Each state change is communicated to the Root Service Node, and by sending e-mails to the contact addresses specified in the Public Key Certificate (PKC) of the Peers.

Initialisation

The startup phase before any incoming connections can be accepted. The VM provides

an HTTP endpoint for health checks to identify when the VM is transitioned into the next state.

Input Phase
The VM waits for all Input Data to be submitted by the Data Custodians. When all slots are filled, processing will begin. If a CTSA grants multiple Data Custodians permission to submit the same Input Data, the VM only accepts the most recently submitted one. The purpose of adding multiple Data Custodians is to allow for redundancy in the event of an unexpected absence.

Calculation
The execution of the main MPC algorithm takes place.

Output Phase
The VM waits for all Data Analysts to retrieve the Output Data. It allows them to download the data only once. When every Peer has retrieved their assigned Output Data, the VM is terminated. Note: If a CTSA grants multiple Data Analysts permission to retrieve the same Output Data, the VM only waits for one. The purpose of adding multiple Data Analysts is to allow for redundancy in the event of an unexpected absence. Internal data sharing shall be solved through a Peer's existing infrastructure.

Cancelled
The CTA or the CT was cancelled, and no Output Data will be available. The reason can be the revocation of the PKC of a Service Node of a Peer, or a request of some Peer or the SO to cancel the CTA or running CT. If the Peers still want to perform a joint computation, they need restart with a new CTSC.

Finished
The CT ran to completion and the CT VMs have been destroyed.

Erased
When the data retention time passes, every Service Node shall erase all data relating to the expired CTA.

4.4.1.2 Contents of the CTSC

The CTSC, visualised in Figure 23, contains the following fields:

Human-readable Title. Unspecified succinct text, allows users to recognise and search for CTAs.

Human-readable Description. Unspecified text which should give non-informed users an overview of what this computation is about, likely containing additional references to external resources.

MPC Engine. Describes what MPC Engine will be used to execute the algorithm. During upload and download, Data Custodians and Data Analysts need to use the matching client side MPC software.

TEE Measurements. Describes the expected measurements of the TEE. This is vendor-specific and thus treated as an opaque field here.

Temporal Specification. A series of deadline and timeout specifications.

Attachments. Any files/documents which can be attached. These may contain legal information or further documentation of the computation context. The System does not process this data.

Computation Parties. A list of CPs which will run the CT. The exact number depends on the MPC Engine used.

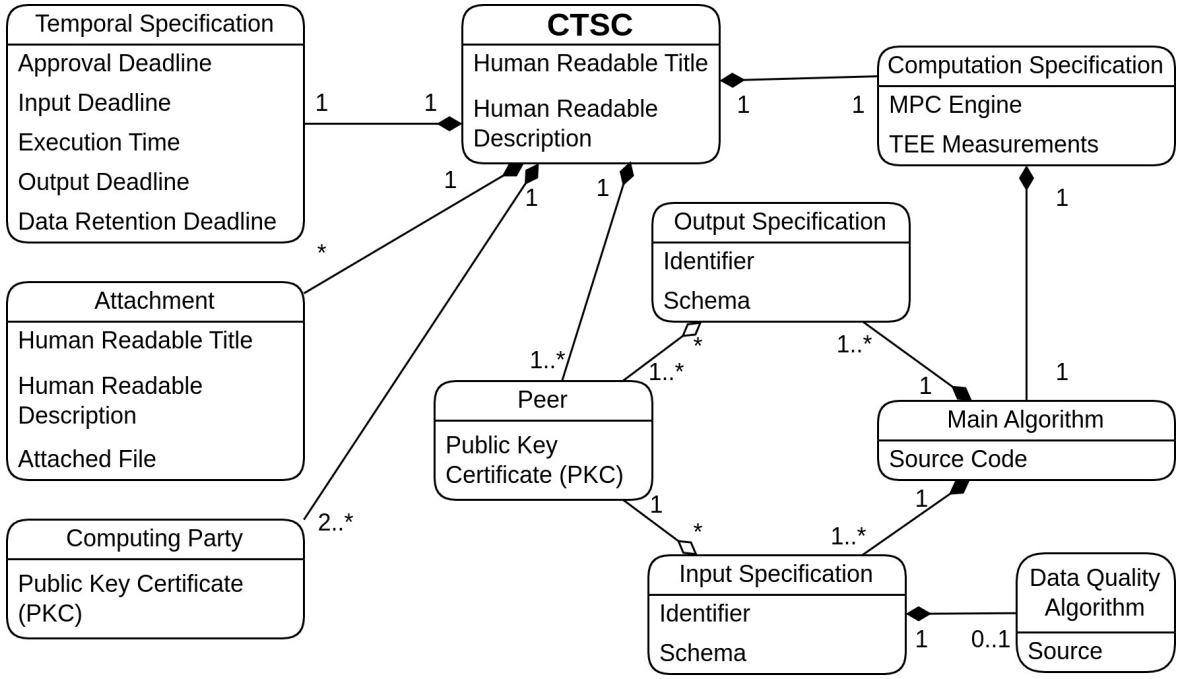


Figure 23. The immutable CTSC contains general information about the CT. Each Peer will augment the CTSC with a signed CTSA to further refine access control (Figure 24)

Peers. List of Clients who participate in the CT. They are identified via the PKC of their Service Node. Peers do not interact directly with the instantiated CT. Instead their responsibility is to further refine the Signatory and Data Custodian roles in the form of the CTSA.

Main Algorithm. The algorithm to be run. Presented as source code, i.e. human-readable sans any obfuscation techniques.

Input Specification. The inputs for the main algorithm. The schema of the Input Data must be explicitly provided to enable the Data Custodians to provide well-formed Input Data without looking at the main algorithm source code. Every Peer gets their own dedicated input specifications. This allows the algorithm to easily identify who provided which data.

Output Specification. The outputs of the main algorithm. The schema of the Output Data must be explicitly provided to let the Data Analysts know what shape the retrieved data will have without looking at the main algorithm source code. The output specification can grant multiple Peers permission to retrieve the same Output Data.

Data Quality Algorithms. Each input specification can be associated with a data quality algorithm which makes sure that the Input Data fulfils certain requirements. The goal is to allow Data Custodians to fix problems with the Input Data before the main algorithm starts. A failed main algorithm cannot be retried; a new CTA must be created instead.

The implementation can choose a non-deterministic data structure or file format to store the CTSC. However, it must display the information in the user interface in a deterministic manner, and it must implement and document a deterministic hashing algorithm.

4.4.1.3 Contents of the CTSA

The CTSA, visualised in Figure 24, contains the following fields:

Peer Identifier. An implementation detail which enables the attribution of the addendum to a

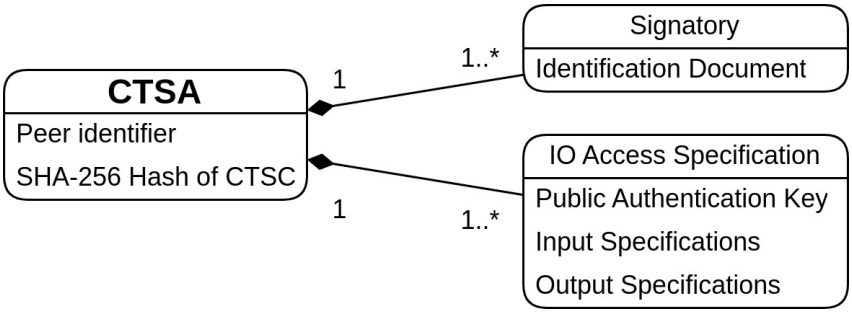


Figure 24. The immutable CTSA refines the individual roles and permissions. Every Peer compiles this information after the CTSC has been created. This way the individual roles and permissions don't need to be exchanged during the creation phase of the CTSC. The *IO Access Specification* determines the permissions for Data Custodians and Data Analysts

specific Peer.

SHA-256 Hash of the CTSC. Ties the addendum to the CTSC.

Signatory Identfication Document. A document which identifies the Signatory or signatories who sign this addendum. Can be, for example, a scan of an ID card, or just a name, depending on the SO's policies. Some matching identifying information must already be present in the SO's member information database.

Public Authentication Key (PAK). A raw public key, i.e. a key with no further meta information or personal information attached. The instantiated CT will use this to authenticate the Data Custodian or Data Analyst. For the purpose of redundancy, such as the unavailability of a specific Data Custodian due to illness, an addendum can list multiple Data Custodians and Data Analysts with overlapping input and output specification assignments.

Input Specifications. Describe what Input Data the Data Custodian can submit to the CT.

Output Specifications. Describe what Output Data the Data Analyst can retrieve from the CT.

The implementation can choose a non-deterministic data structure or file format to store the CTSA. However, it must display the information in the user interface in a deterministic manner, and it must implement and document a deterministic hashing algorithm.

4.4.1.4 Contents of the CTA

The CTA, visualised in Figure 25, contains the following fields:

Universally Unique Identifier. The ID for the CTA. This must be a version 4 UUID, meaning a 128 bit number with 122 bits of randomness and 6 bits of version information.

Signatory Identifier. An implementation detail which allows the signature to be attributed to a specific Signatory listed in the CTSA.

Signature Document. A document containing the signature. Could be something electronic like *ASiC-E*, or a scanned document, depending on the SO's policies.

Digital Signature. Cryptographic signature over the CTSA, given by the Task Organiser using the Asymmetric Signing Key Pair of their Service Node.

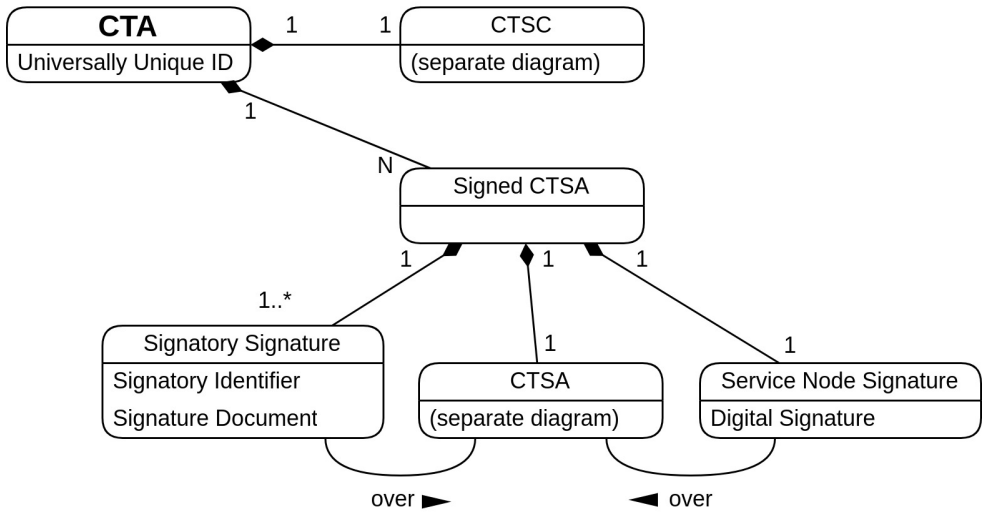


Figure 25. The CTA comprises the CTSC (Figure 23), CTSAs (Figure 24) and signatures

4.4.1.5 Access

The primary data store for CT Artefacts is at the Root Service Node. Intermediate Service Nodes receive updates to CT Artefacts through their respective parent Service Node, using the PKC for authentication purposes.

4.4.2 Service Node Certificates

Every Service Node is identified by its Asymmetric Signing Key Pair. The public key is signed by the Key Pair of the parent Service Node. The signature is stored together with the public key as a PKC, thus forming a certificate hierarchy (Figure 26). It is strongly recommended that the SO uses HSMs: the root key pair must be stored in an offline HSM, and the Asymmetric Signing Key Pair of the Root Service Node must be stored in an online HSM (Figure 29). This is to avoid the high overhead of creating new PKCs for all connected Service Nodes when one of the SO's key pairs is leaked. This is less of an issue for other members; however, use of HSMs is still recommended.

4.4.2.1 Meta Information Contained in the PKC

The PKC must contain non-sensitive contact information regarding the Service Node. If possible, it must not contain sensitive information.

1. Company Name
2. Department Name
3. Non-personal Contact E-Mail Address

4.4.2.2 Uses of the PKC

The PKC and the certificate chain serve the following purposes:

Authentication when Joining the System
: The IT Administrator or General Manager configures the PKC of the Root Service Node (Figure 29) when first deploying a Service Node. The Service Node will then verify that the PKC of

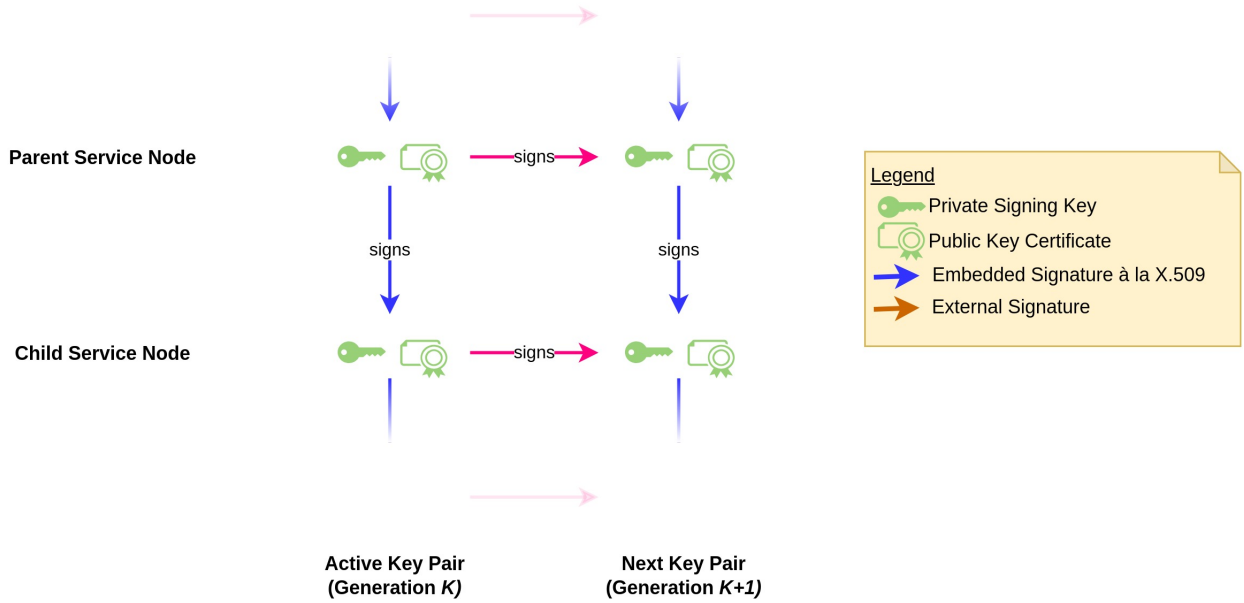


Figure 26. Service Nodes rotate their keys automatically under normal circumstances. The rotation frequency is determined by the Root Service Node. When rotating keys, the new PKC is signed with the previous key pair. This signature is used by the parent and child Service Nodes to attest the authenticity of the new PKC

their parent node is transitively signed by the PKC of the Root Service Node. This eliminates the possibility to connect to a parent Service Nodes which is not part of the System.

Quick Overview of Peer Identities
The organisation and department details can be extracted easily from the PKCs in a CTSC, thus providing a quick overview of participating companies.

Non-personal Permission Proxy Within a CTSC (signing the CTSA)
The CTSC contains the PKC of each Peer. Task Organisers of each Peer, who have the permission to use the respective PKC of their Service Node, can then assign granular upload and download permissions to specific Data Custodians by signing their PAK (Section 4.4.3) as a part of the CTSA. Newly created CTAs should use the *active* PKC, but signing an existing CTSA can be done with a *passive* PKC (see Section 4.4.2.3 for the lifecycle of PKCs).

Asynchronous Communication when CT is Ready for Upload or Download
As a means of asynchronous communication, a Service Node receiving telemetry data could send an e-mail to the Peers of a CT to inform them when data can be uploaded or downloaded. The e-mail address of a Peer is stored in its PKC. This e-mail address should be a non-personal mailing list address, e.g. `joconde@dpt1.example.com`.

Authentication when Connecting to the Parent Service Node
Task Organisers and Signatories use the Key Pair to receive updated CT Artefacts and telemetry through their parent Service Node, as described in Section 4.4.1.5.

4.4.2.3 Lifecycle

The lifecycle stages of the certificate tree are as follows (Figures 26 to 28):

Root PKC Creation (Figure 29)
The root key pair should ideally not change. It serves as a trust anchor for new nodes and is essentially the identity of a given System instance. Hence it is recommended to use an offline

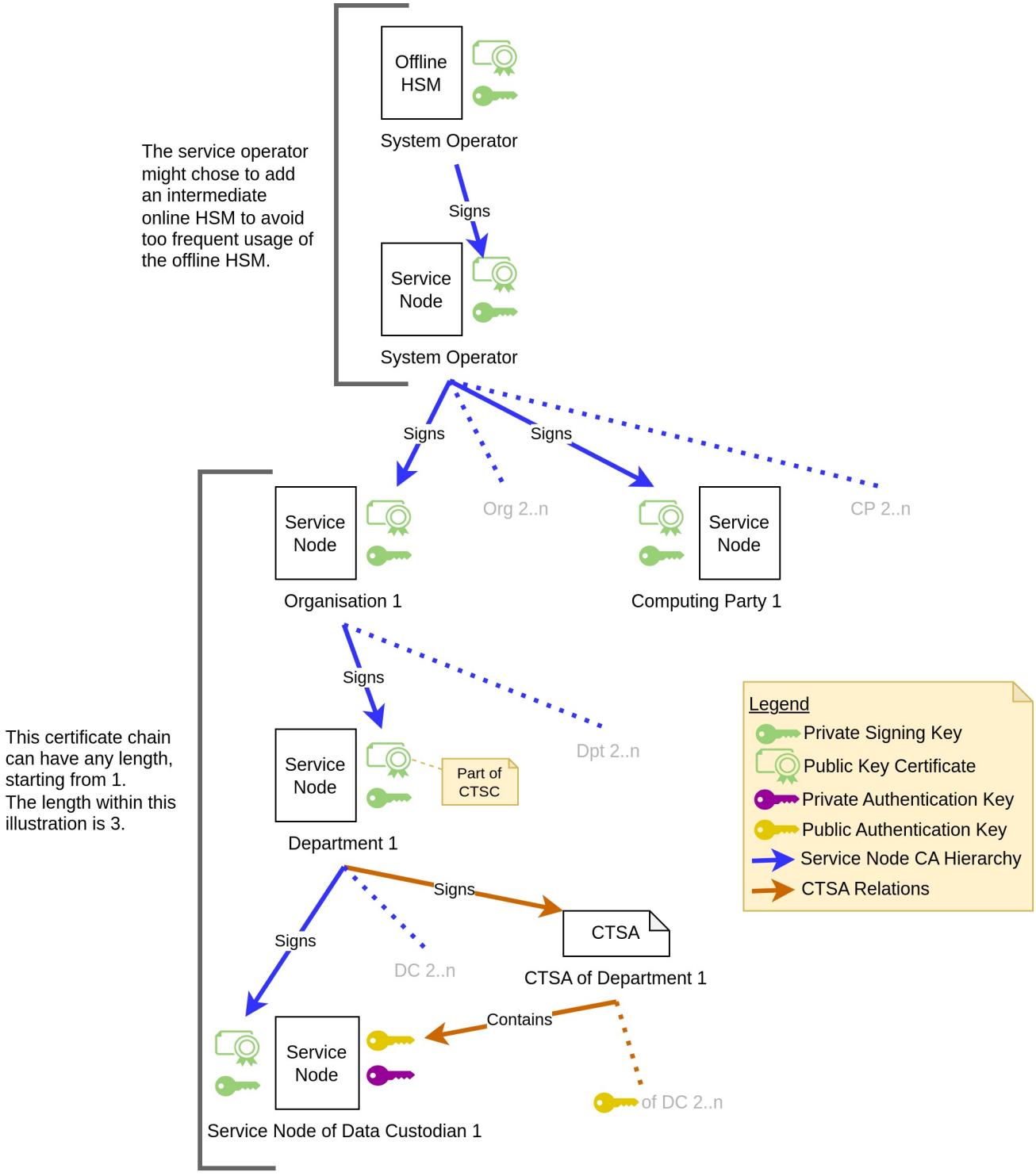


Figure 27. Connection of the PKC, CTSC, CTSA and PAK

HSM for this.
When the key pair of the Root Service Node is rotated, the root key pair in the offline HSM needs to sign this. To avoid frequent usage of the offline HSM, the SO can use an intermediate key pair which is stored in an online HSM.

Root Service Node PKC Creation (Figure 29)
This key pair should be stored in an online HSM. A revocation of this key requires manual communication with all members and should be avoided if possible.

Member Service Node PKC Creation (Figure 30)
Members' key pairs are not necessarily required to be stored in an HSM. The initial PKC needs to be manually registered at the Service Node of the parent. Further regular key rotations happen automatically.

Automatic Key Pair Rotation (Figure 31)
Key pairs are automatically rotated. The rotation frequency is dictated by the Root Service Node. The recommended expiration time is 24 months, but a key rotation should be carried out every 6 months. This means a PKC is *active* for 6 months and *passive* for 18 months. The 18-month expiration time of a passive PKC can be decreased, but it must be higher than the longest anticipated CTA lifetime (from *Emerging* to *Finished*, Figure 21). Otherwise a Task Organiser may be unable to query updated CT Artefacts.

Key Pair Revocation (Figures 32 and 33)
An IT Administrator or General Manager can revoke a Key Pair using the Service Node if they suspect that the private key has leaked. The revocation request is sent to the Root Service Node, and it must be signed with the presumably leaked private key. All CTs where this PKC is used for a Peer must be cancelled, and this Service Node can no longer access the respective CT Artefacts. If the PKC appears only within the certificate chain as an intermediate certificate, then cancellation is not required. Additionally, all IT Administrators or General Managers of the child Service Nodes must create a new PKC manually (Figure 30). Descendant Service Nodes can still use their PKC for the access of CT Artefacts, even if some intermediate certificate was revoked, since the access only verifies signatures but not the chain validity.

Physical Loss of the Key Pair
A physically lost key cannot be revoked by the (once) owning Service Node, as the revocation request cannot be signed with the private key. An IT Administrator of the Root Service Node can, however, manually mark the certificate as revoked to cancel any active CTA associated with that Key Pair. Note that the Service Node with the lost Key Pair can no longer query the associated CT Artefacts from the Root Service Node.

4.4.3 Data Custodian and Analyst Asymmetric Authentication Key Pair

Data Custodians and Data Analysts use raw asymmetric keys without the additional meta information which is part of the PKC: the access control information associated with the keys are part of the CTSA, and the name of the owner is sensitive data which is not required for the operation of the System.

4.4.3.1 Uses of the Asymmetric Authentication Key Pair

The Asymmetric Authentication Key Pair serves the following purposes:

Authentication when Interacting with the CT
The PAK is used during remote attestation of the CT VM for authentication. The CT VM recog-

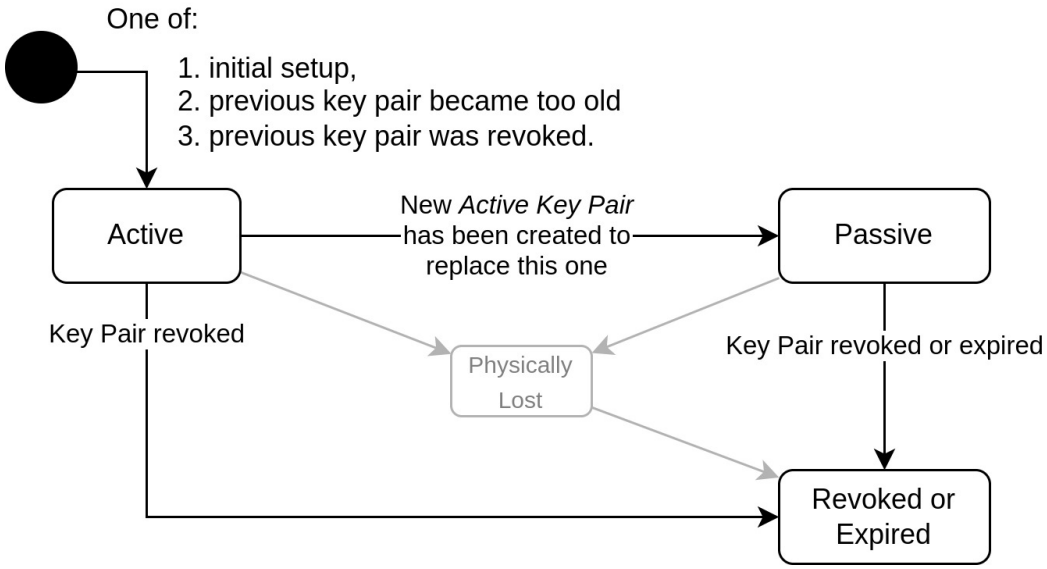


Figure 28. The lifecycle of a Asymmetric Signing Key Pair

nises the PAK from one of the CTSAs and grants them permissions to submit Input Data or retrieve Output Data accordingly.

Lookup of CT Artefacts
Data Custodians and Data Analysts use the Key Pair to query new CT Artefacts from the Root Service Node, as described in Section 4.4.1.5.

4.4.3.2 Lifecycle

The lifecycle stages of the Asymmetric Authentication Key Pair are as follows (Figures 34 and 35).

Creation
The Task Organiser sends the UUID of the CT to the Data Custodian (or Data Analyst, omitted below for brevity). This is done outside of the System, as it is expected that the Task Organiser and Data Custodian already have an established means of communication, e.g. corporate e-mail or project management tools. The Data Custodian uses the UUID in their Service Node's user interface to create a new Asymmetric Authentication Key Pair and send the PAK to the Task Organiser's ancestor Service Node. At this stage, the PAK is *not* sent to the Root Service Node.

Review
The Task Organiser sees the new PAK and the user identifier, e.g. LDAP name which the Data Custodian uses to log into their Service Node.

Commit
They can now assign permissions to the PAK, which is stored as a part of the CTSA (Figure 24).

Data Retention
The Service Nodes of the Data Custodian stores the Asymmetric Authentication Key Pair, and intermediate Service Nodes store the PAK as a part of the CTSA. Both are deleted upon the expiry of the CTA data retention time.

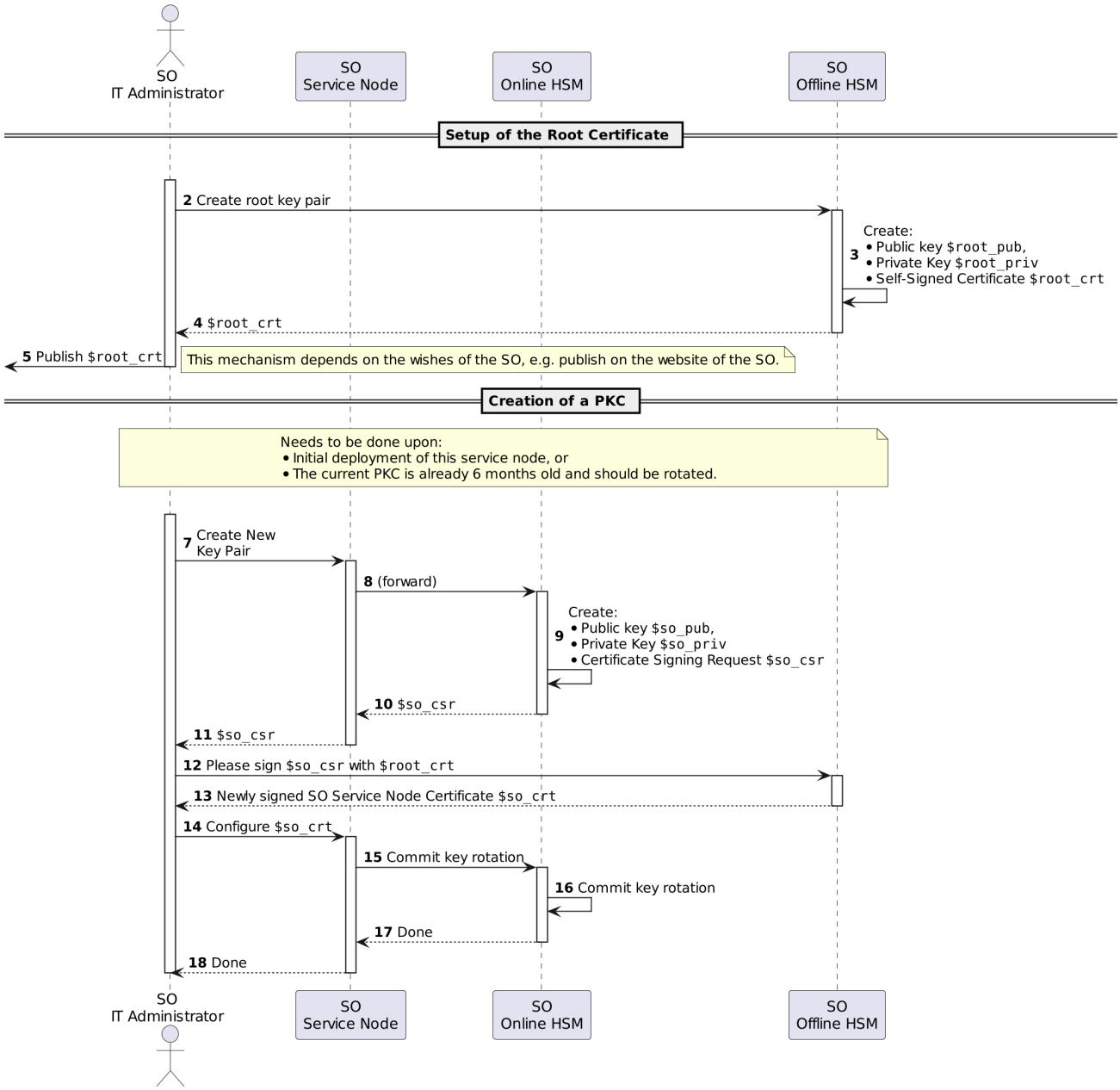


Figure 29. A sequence diagram which shows how an IT Administrator or General Manager of the SO creates the root PKC and the PKC of the Root Service Node

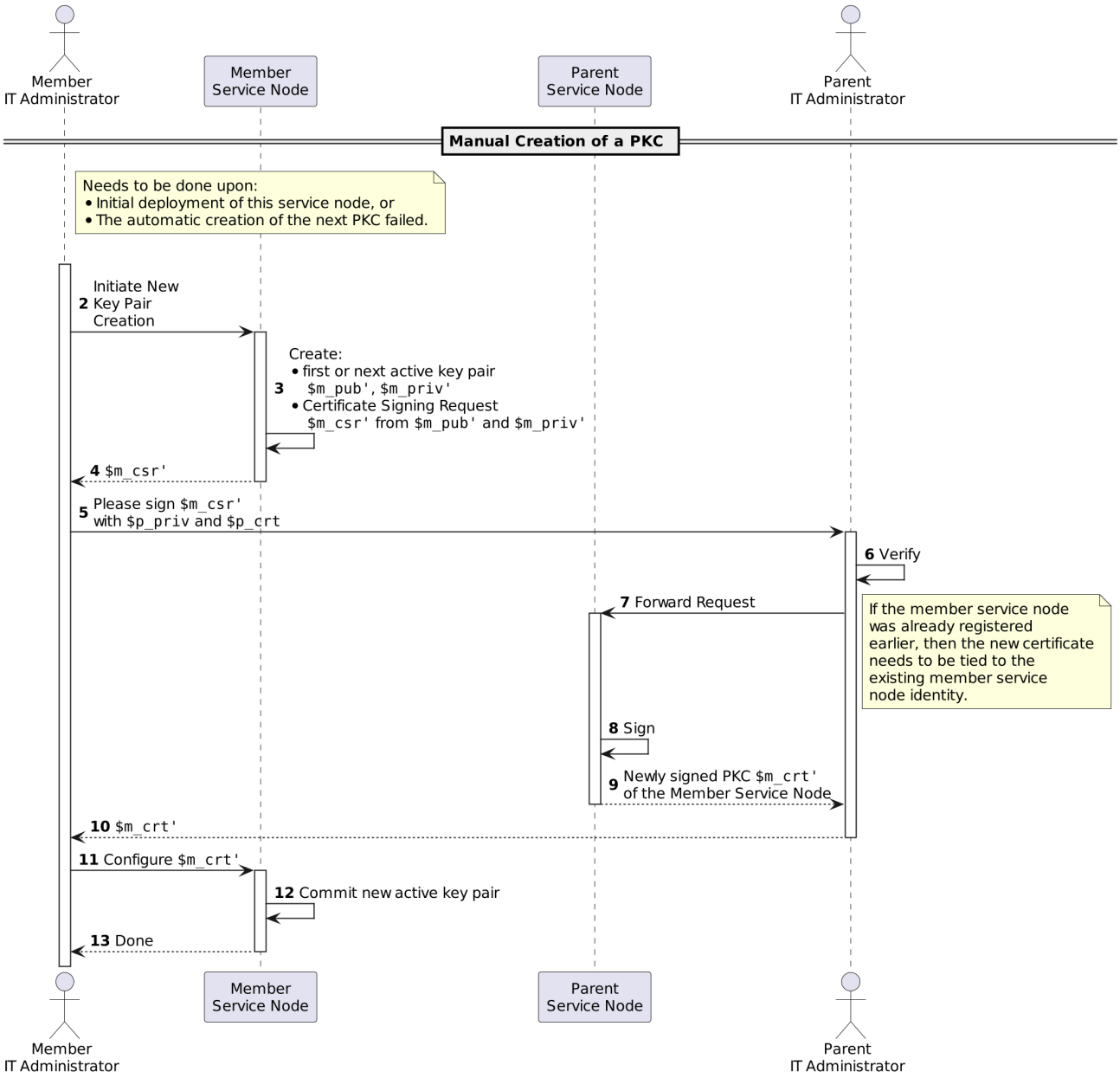


Figure 30. A sequence diagram which shows how an administrator of a Member manually creates a PKC. This is necessary when the automatic rotation of the PKC fails (Figure 31)

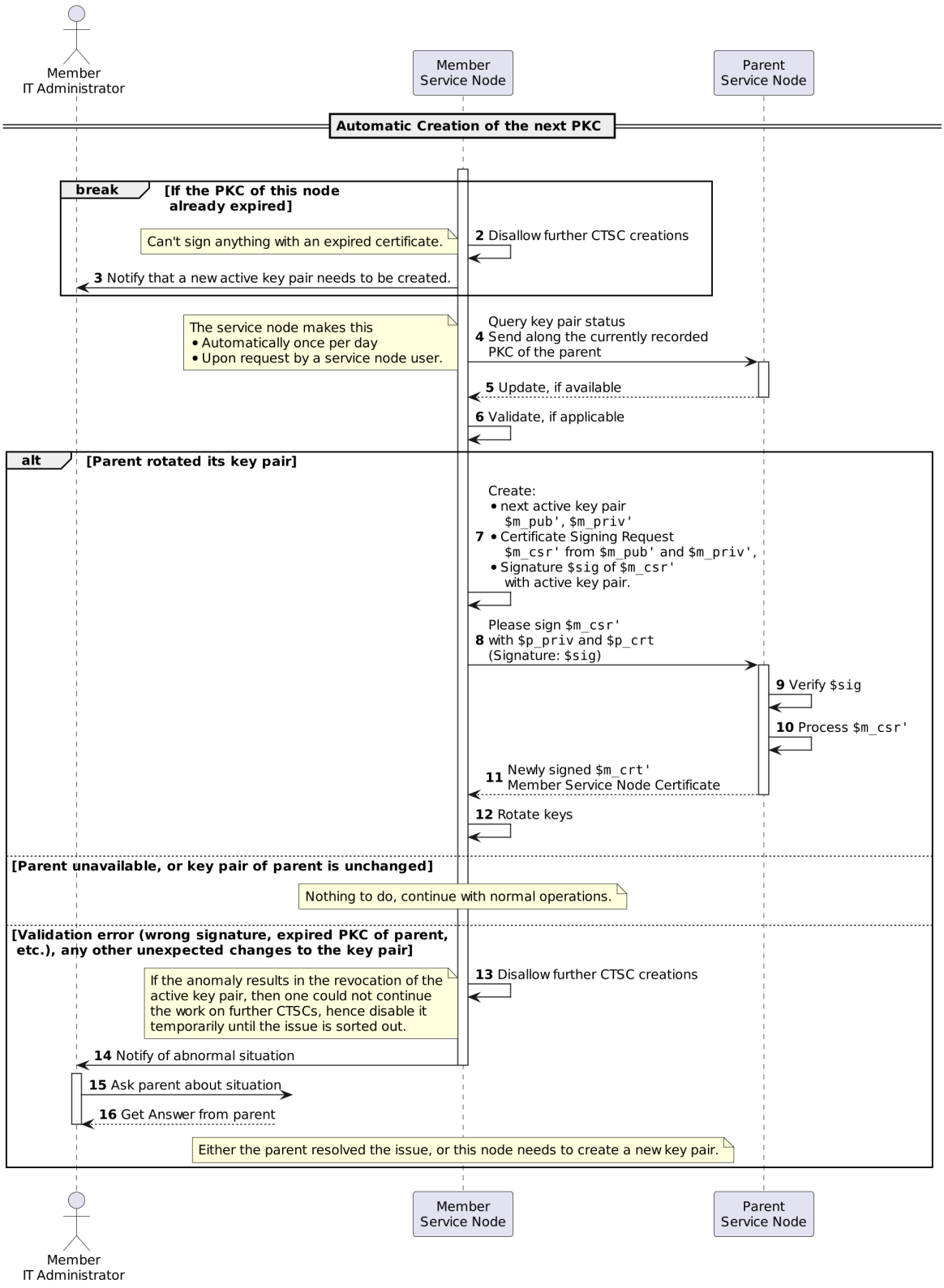


Figure 31. During regular operations the Service Nodes of members should automatically rotate the active key pair and PKC. Error conditions require intervention of the administrator (Figure 30)

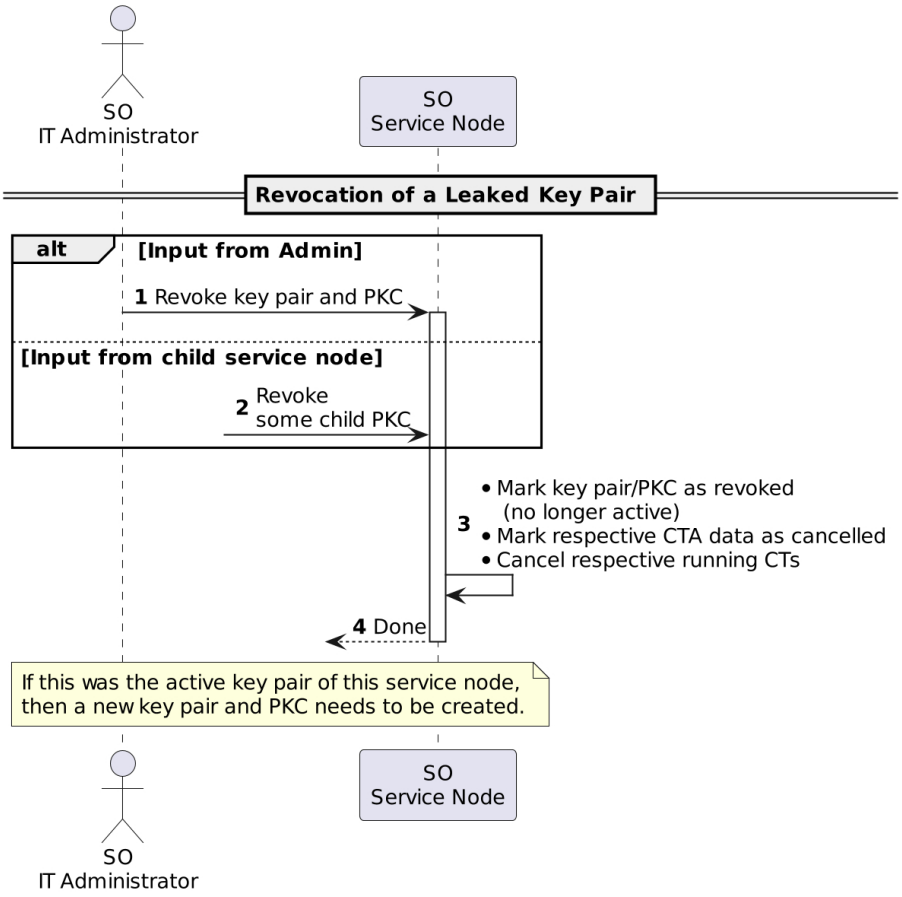


Figure 32. When a PKC is revoked, pending and instantiated CTs which contain this PKC are cancelled (Figure 21). The request might come from a child Service Node, see Figure 33

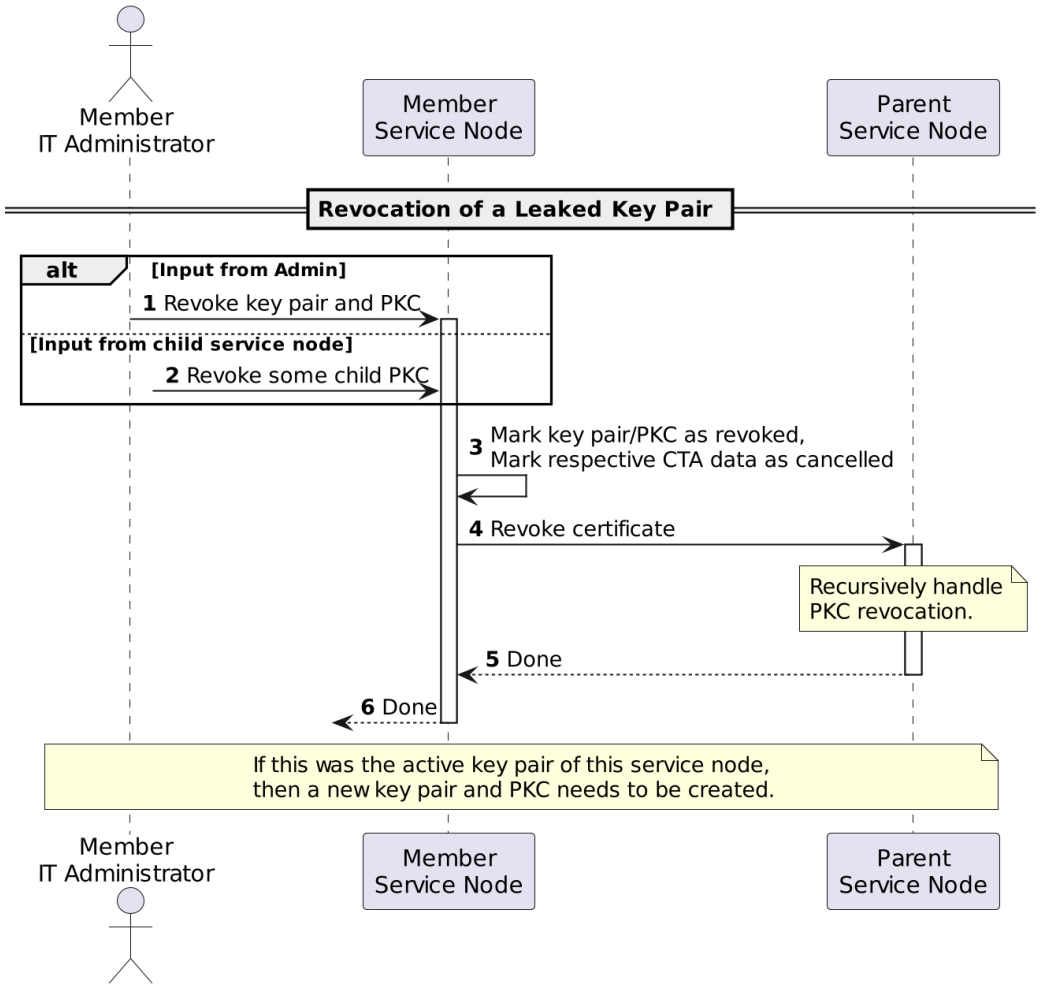


Figure 33. When a PKC is revoked, pending and instantiated CTs which contain this PKC are cancelled (Figure 21) and the revocation request is sent upwards to the parent Service Node. Child Service Nodes will detect the revocation of the PKC of their parent when they attempt to automatically create their own next PKC (Figure 31)

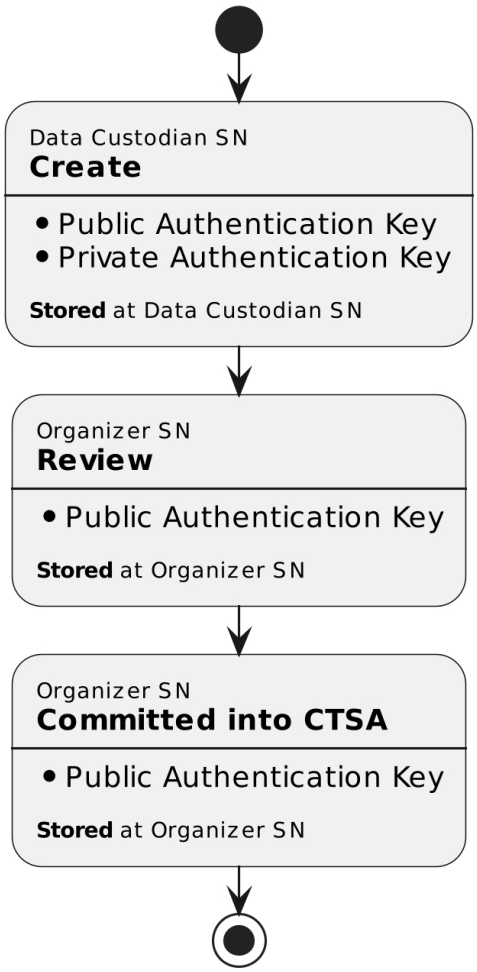


Figure 34. The lifecycle of a Asymmetric Authentication Key Pair of a Data Custodian (or Data Analyst)

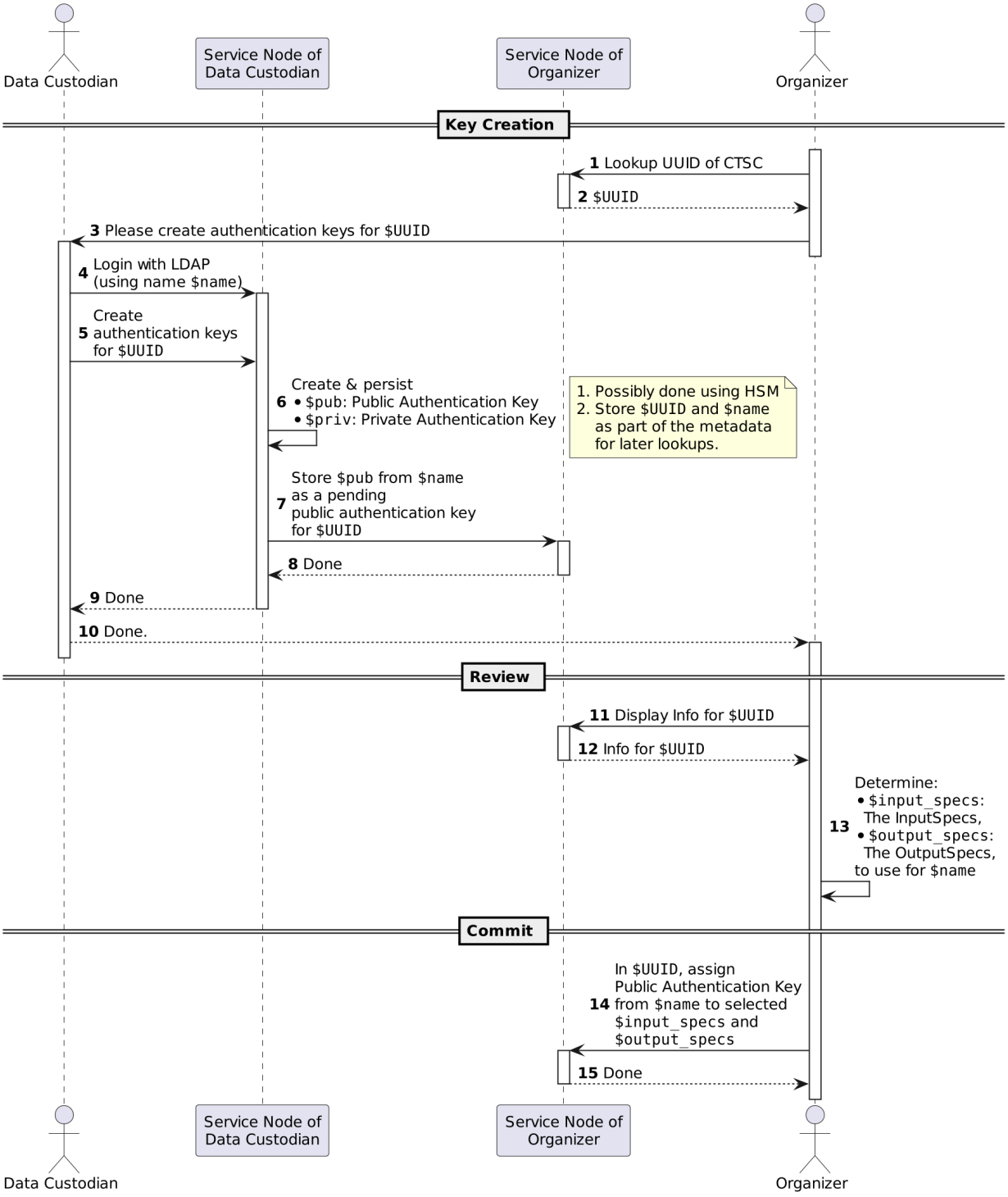


Figure 35. This sequence diagram shows how a Data Custodian (or Data Analyst) creates a Asymmetric Authentication Key Pair, and how it reaches the CTSA

4.4.4 Signatory Certificate and Signatures

The Signatory needs to legally sign the CTSA to indicate their consent that the Peer accepts all the data inside of the CTSC (Figure 23) and the CTSA of this Peer (Figure 24). They use an identity solution which is independent of the System, e.g. physical ID cards or digital signatures, depending on the identification solution which the signatory has access to. Specifically, this means the System accepts anything as a signatory certificate, e.g. a JPG file or a X.509 certificate in the PEM format. Thus the format of the signature is equally not further specified. The SO needs to manually verify the certificate and signatures. It could be of any form, e.g. a JPG scan of a physical ID card, or an electronic certificate. It is stored as part of the Member's information stored at the SO

4.4.4.1 Uses of the Certificate and Signatures

Matching the Signatory Signature with the Signatory Identity
: The SO uses the signatory certificate to identify who gave the signature over the CTSA, and matches it with existing Member's information.

Legally Signing the CTSA
: To ensure that the Peer legally accepts to participate in the CT.

4.4.4.2 Lifecycle

The signatory document is issued by an external identity and has its own lifecycle independent of the System. However, digital copies of the signatory certificate and the signatory signatures are stored as follows:

As a part of the Member's information stored at the SO
: The signatory certificate needs to be stored as least until the Member leaves the System.

As a part of the CTSAs
: The signatory certificate and signatory signature are stored at related Service Nodes at least until the expiration of the data retention time of the respective CTSC.

4.4.5 Local User Management System

The Service Nodes are dependent on a local user management system like LDAP to identify users and their permissions. More specifically, organisers and signatories are managed as groups, such that the creation of CT Artefacts of a Peer can be driven forward by multiple individuals.

A special note for the situation where a Member creates sub-Service Nodes, like in Figure 27: without the usage of *JSON Web Signatures* or similar technologies, the parent Service Node has no technical measure to ensure that a child Service Node enforces the same LDAP rules. In this case, the correct setup of the user management should be inspected when the child Service Node joins the System, i.e. it has to be covered by regular enterprise policies. Further, local policy rules can prevent wrong behavior. For example, if a Data Custodian manages local Service Node by themselves, its parent Service Node should add a policy to reject any CTSC from the Service Node of the Data Custodian, as the Data Custodian could easily make themselves a Task Organiser on their own node. Again, this situation can be avoided when using *JSON Web Signatures* or similar technologies.

4.4.6 VM Address

Each CT VM exposes endpoints for remote attestation and for communication, used by the Data Custodians, Data Analysts and the other CT VMs of the same CT. When a CT VM is created, the Service Node of the CP sends the addresses of the endpoints to the Root Service Node. They are CT Artefacts and exist while the CTA is in any of the states from *Instantiated* to *Erased* (Figure 21).

4.4.7 Logs & Telemetry – Service Node Logs

Every Service Node logs incoming and outgoing requests, errors, and further noteworthy information. The logs must be generated in a structured manner and be compatible with widely used tools, e.g. using the `syslog` protocol[11].

4.4.7.1 Contents

The log contains at least the following information:

- For each request, connection information of the other Service Node (IP address, common name from the certificate, serial number of the certificate);
- CT Artefacts;
- PKC rotation data;
- Information on addition or removal of Service Nodes;
- Login information (user name, IP address).

4.4.7.2 Uses

The sole consumer of the logs is the Auditor who uses them to proactively and retroactively identify attacks. Other stakeholders should not need to look at the logs.

4.4.7.3 Lifecycle

Logs are generated when a request arrives, when errors occur, and for further situations at the discretion of the System implementer. Logs are stored for as long as required by applicable laws and specified in the System Agreement.

4.4.7.4 Access

Access to the logs shall be restricted to a dedicated auditor group. Auditors of some ancestor Service Node shall be granted access upon request to the logs which were produced by descendant Service Nodes. The Auditor could be granted access to the log storage system itself, or be sent an archive file containing the logs.

Logs are not cryptographically protected by the System. It is expected that the external log storage has its own protection measures (possibly cryptographic) to provide integrity and non-repudiation, and is akin to a write-once store, except for data retention functionality. Further, most logs are present in a parent and a child Service Node.

[11] The Syslog Protocol `https://www.rfc-editor.org/rfc/rfc5424` Last accessed: March 2025

4.4.8 Logs & Telemetry – CT VM Logs

Every CT VM logs redacted incoming and outgoing requests and `printf` messages produced by the MPC algorithms.

4.4.8.1 Contents

The log contains at least the following information:

- The CTA to identify the CT within this CT VM;
- The redacted requests and the identifying information of Data Custodians and Data Analysts (IP address, PAK or some pointer to the PAK within the CTA);
- State transition information (see Section 4.4.1.1 under *Instantiated*);
- `printf` messages created by the MPC algorithms.

4.4.8.2 Uses

The primary consumer of the logs is the Auditor who can use them to proactively and retroactively identify attacks. Other stakeholders should not need to look at the logs during regular use. An exception is during the development of the Main Algorithm where the developer requires log access to debug output messages.

4.4.8.3 Lifecycle

Log entries are created within the CT VM while it is in the *Instantiated* state (Figure 21), and sent to a telemetry collection service outside the CT VM. When the CT VM terminates, a copy of the log should be stored within the same place as the other logs created by Service Nodes of the CP (Section 4.4.7). The CT VM logs are stored for as long as required by applicable laws and specified in the System Agreement.

4.4.8.4 Access

Access to the logs shall be restricted to a dedicated auditor group. The Auditors shall be granted access to the logs upon request. They could be granted access to the log storage system itself, or be sent an archive file containing the logs.

Logs are not cryptographically protected by the System. It is expected that the external log storage has its own protection measures (possibly cryptographic) to provide integrity and non-repudiation, and is akin to a write-once store, except for data retention functionality.

4.4.9 Logs & Telemetry – CT VM State Change Information

An *Instantiated* CT VM goes through multiple phases (see Section 4.4.1.1 under *Instantiated*). When the CT VM advances to the next state, or when it terminates due to an error, it will publish the information as telemetry.

4.4.9.1 Uses

Information about state changes or errors is required for the Data Custodians and Data Analysts.

4.4.9.2 Lifecycle

State change Telemetry Aggregator is available as long as the data retention time of the CTA.

4.4.9.3 Access

The access to and further processing of Telemetry Aggregator data is configured individually per Service Node.

5 Quality Properties

This chapter explains how the System meets various quality properties. To provide a structured analysis, we utilise architectural perspectives [3]. Specifically, this iteration examines the System from the security and usability perspectives. For each perspective, we describe how the relevant requirements established in *D1.1 Usage Scenarios and System Requirements* [1] are fulfilled.

5.1 Security Perspective

The System has multiple security requirements. These requirements cannot be addressed within a single view as they span across multiple views. In this section, we define the model used to identify threats in the System. After that, we explain how the System mitigates threats, i.e. how the System meets the security requirements.

5.1.1 Threat model

The threat model for the System is defined as an attack tree. Although availability and integrity of the System are important, this threat model focuses on the confidentiality of Restricted Data. The ultimate goal for an attacker, as represented by the root of the attack tree (shown in Figure 36), is to breach the confidentiality of a Client's Restricted Data. This approach has been effective in identifying potential threats to the System and in laying out measures to increase the cost and reduce the likelihood of these attacks. Some protection measures detailed in Section 5.1.2 are contextualised using snippets of deeper layers of the tree. The identified threats will be used in subsequent risk analysis in the next versions of this of this document.

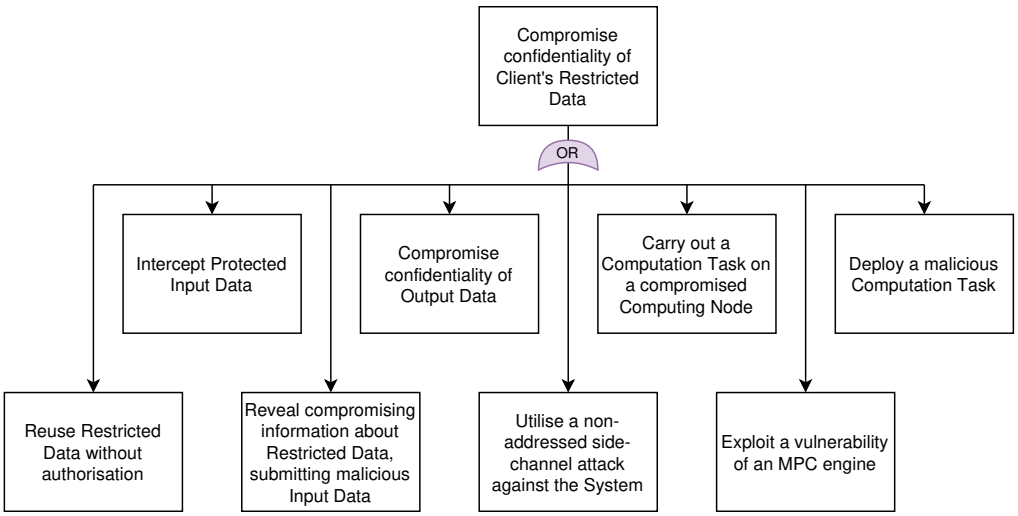


Figure 36. First layer of the attack tree describing the goal of compromising confidentiality of Client's Restricted Data. Accomplishing any of the sub-nodes is sufficient to achieve the goal (OR gates)

5.1.2 Security controls and measures

Table 1. Security requirements

SYS-1.3	The System shall be robust to intrusions at up to two out of three Computing Nodes.

How met:

1. According to the data plane modularity principle (PR-6), this can be fulfilled if the external MPC Engine chosen for a CT is robust to intrusions at up to two out of three Computing Node.
2. The MPC-over-TEE model (see ADR-1) provides enhanced security guarantees to MPC protocols [2], which may be argued to make this requirement easier to achieve.
3. The System distributes a CTA to each involved Computing Node separately (as described in ADR-4), the Computation Task Orchestrator in each Computing Nodes follows the CTA policy independently. If two Computing Nodes deviate from the CTA to use Protected Data outside of the prescribed context, the third Computing Node would not comply.

SYS-2.1	The System shall enable Users to verify the identities of other Members contained in the Computation Task Specification locally. The verification shall not rely on trust in the System Operator.

How met:

1. All Clients have a local Node Identity Manager component which they use to maintain their own trusted view of the Service Nodes (as described in ADR-5). The component persists Peer certificates and may refuse to proceed with the CTA consolidation if trust is not established between the Clients.
2. The Node Identity Manager component is a part of the Client Software that is to be deployed on the Client's premises, which is out of direct control of the SO. Thus, trust in the SO is not required.
3. The Client Software is open-source. Clients are encouraged to review the source code of the software before deployment.
4. Trust establishment is based on out-of-band communication via channels outside of the System (as described in ADR-5).

The attack tree in Figure 37 shows how an attacker can impersonate a Member – setting the context of this requirement. Impersonation involves convincing a User to enter a CT with a malicious Member using a falsified identity, putting Restricted Data at risk. The figure depicts the applied measures shown as pink annotations. The descendant nodes of the security measures show the attack strategy to circumvent them. It should be apparent that the likelihood of the post-measure steps greatly diminishes the risk of an impersonation attack.

Table 1. Security requirements *(continued)*

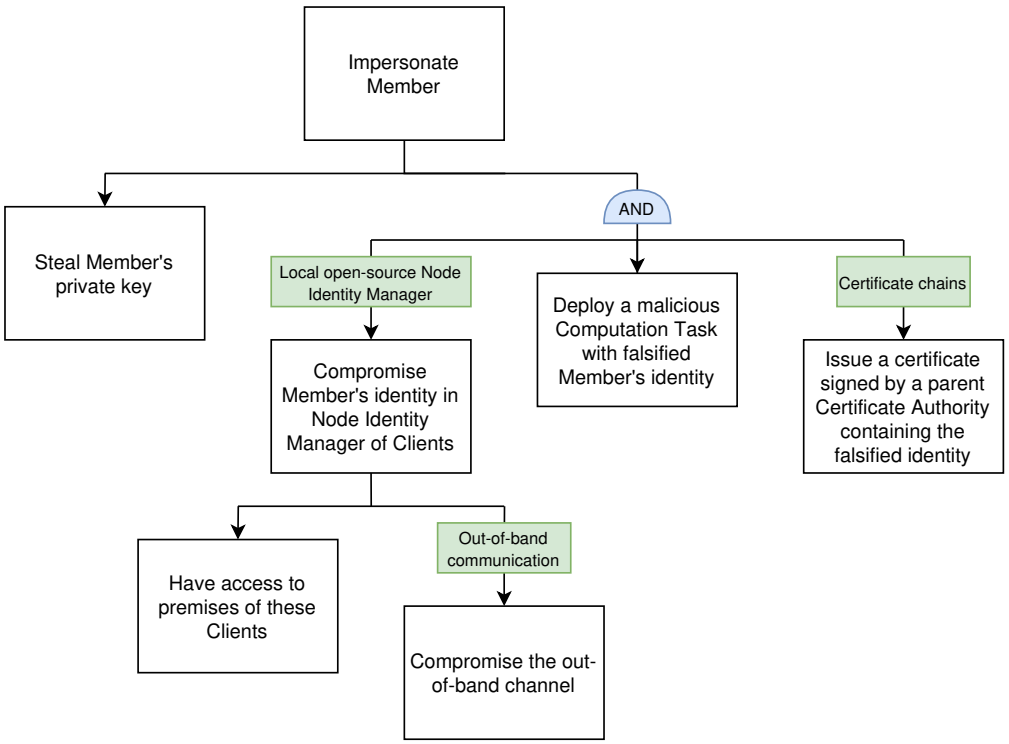


Figure 37. Attack sub-tree for Member impersonation. The AND node specifies that all descendants must be executed for the goal (root) to be fulfilled.

SYS-2.3	The System shall allow Input and Output parties to verify the integrity of the copies of their Computation Task Agreements deployed in Computing Nodes.

How met:

1. Each Computing Node executes a CT in a Trusted Execution Environment (see ADR-1).
2. TEEs enable the Clients to verify the integrity of the deployed CTA and the MPC Engine using the remote attestation procedure.
3. The Client Software's IO Controller component runs the remote attestation procedure for each involved CT VM.

Table 1. Security requirements *(continued)*

SYS-2.2	The System shall use a Computation Task Specification approval mechanism that provides integrity and non-repudiation.

How met:

Integrity of approval by a Client

1. The CTSA is digitally signed by the Client Node that the CTSC designates. This is marks the approval of the Client's Task Organiser who authorises it. The CTSA is further signed by the Signatory.
2. Either provides integrity for the significant part of the CTS as the signed CTSA contains the hash of the CTSC.

Integrity of approval by the SO

3. The SO forwards the CTA to the Computing Nodes over an authenticated secure channel only after they have approved it.

Non-repudiation of approval by a Client

4. The CTSA can only be signed by a designated Client Node PKC. A Client might claim that the PKC in the CTSC did not belong to them – its sufficient to verify that the certificate chain includes the PKC (or its successor) that was initially attributed to the Client (legal entity) during the on-boarding procedure. Claiming that the approval was given with a leaked private key assumes that they have initiated PKC revocation before the fact.
5. The signature provided by the Signatory makes it more difficult to repudiate the approval by a Client, as the CTSA includes the identity of the Signatory, who is bound to the legal entity of the Client.

Non-repudiation of approval by the SO

6. Approval by the SO is verifiable via the logs of the independent Computing Nodes, which witness the receipt of the CTA.

SYS-4.1	The System shall provide the System Operator with Identity and Access Management (IAM) for Member management and access.

How met:

1. Each Service Node has a local Node Identity Manager component. The stored identities can be associated with a set of permissions by the Role and Access Manager, enabling the SO to manage Members and their access to the System.
2. As the System utilises a tree topology (as described in ADR-4), interorganisational communication gives SO the power to intercept and enforce policy on every exchange. Thus, the SO is able to centrally manage Members' permissions.

SYS-5.2	Computing Nodes shall only accept and enforce Computation Task Agreements sourced by trusted means.

How met:

1. The prerequisites for secure channel establishment are certificate chains issued for each Service Node (described in ADR-5) and established trust between Service Nodes in a CT.
2. The Computing Nodes shall accept CTAs from the SO's Service Node only if the sender and recipient are mutually authenticated utilising the mTLS protocol.

Table 1. Security requirements *(continued)*

SYS-6.1	The System shall associate all data with its respective Computation Task to enforce data lifecycle policies.

How met:

1. All Input Data, Output Data and any intermediate data of a CTs are bound to Slots, which are owned by the CT VM, unusable outside of the CT VM.
2. The Task State Machine actively manages the creation and deletion of the Slots w.r.t task state.
3. Upon a sudden crash of the CT VM, any orphaned data is left unusable (see ADR-1).

SYS-6.2	The Computation Task shall expect time deadlines for task lifecycle stages, specifically manual I/O operations including Protected Input Data upload and Protected Output Data retention.

How met:

1. The CTA contains fields such as *Protected Input Data upload deadline* and *Protected Output Data retention deadline*. These deadlines are enforced by the Task State Machine of each CT VM.

Table 1. Security requirements *(continued)*

SYS-6.5	The System shall employ technical measures ensuring that all Protected Input Data and Interim Data are permanently and securely deleted or rendered permanently illegible immediately after the completion of the Computation Task or upon reaching the deadline.
SYS-6.6	The System shall employ technical measures ensuring that all Protected Output Data are permanently and securely deleted or rendered permanently illegible immediately after retrieval by all Output Parties or upon reaching the deadline.

How met:
Data deletion guarantees provided by TEEs are not absolute. Confidential data written to a disk is encrypted with a hardware-protected sealing key. However, if an unauthorised copy of this encrypted data is created outside the TEE, it may become subject to future vulnerabilities, e.g. if a future attack is found that breaks the encryption implementation used by the TEE. Note, however, that thanks to the MPC layer, such a future attack would be successful only if unauthorised copies are made of all data from all computing parties (*D2.1 Technology Survey and Analysis* [2]). This yields guarantees that are just as strong as storing the data in encrypted memory – both provide for confidentiality and integrity as long as hardware encryption is not broken.

1. Each Computing Node executes the CT in a Trusted Execution Environment (see ADR-1). The TEE ensures that once the VM is destroyed, all of the Protected Data (including Protected Input Data, Protected Interim Data, and Protected Output Data) that was locally stored are effectively destroyed (i.e. rendered permanently illegible). This is guaranteed, as with shutdown of the TEE's VM, the decryption key is destroyed.
2. Each Computing Node has a local Task State Machine component that ensures that the specified expiration times are met. When the expiration time is reached, the Task State Machine initiates the deletion of locally stored Protected Data and the VM.
3. The Task State Machine component initiates the deletion of the locally stored Protected Data and the VM after Output Data retrieval by all OPs.
4. The Task State Machine component initiates the deletion of the locally stored Protected Input Data and Protected Interim Data after the completion of the CT.
5. If the external MPC Engine chosen for a CT is robust to intrusion at up to two out of three Computing Nodes, all Protected Input Data, Interim Protected Data and Protected Output Data is guaranteed to be removed as long as at least one of of three Computing Nodes remains honest.

The attack tree in Figure 38 shows how an attacker can reuse Restricted Data without authorisation when the said measures are applied. The figure also depicts the context of the applied measures as well as the residual risks.

Table 1. Security requirements *(continued)*

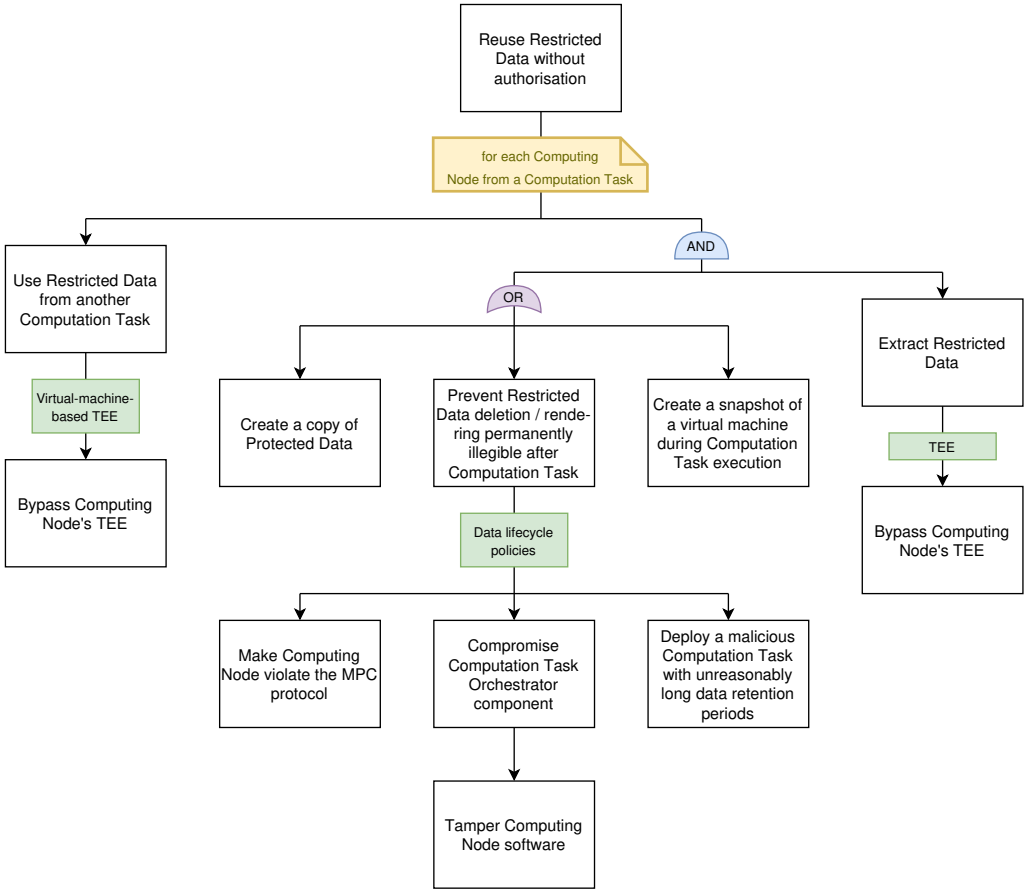


Figure 38. Attack tree for unauthorised Restricted Data reuse

SYS-9.1	The System shall not disclose Input Data values to anyone other than the Input Party, unless explicitly stated otherwise in the Computation Task Specification.
SYS-9.2	No single entity shall have the ability to learn any Input Data values of any individual computation task, unless explicitly stated in the Computation Task Specification.

How met:

1. Each Computing Node executes a CT in a Trusted Execution Environment (see ADR-1). The TEE prevents executed code tampering, data tampering, and data extraction (including data or VM snapshotting).
2. The System does not provide an interface that would enable any Interim Data to be downloaded or queried. The System enables querying only the main data analysis algorithm's output that is explicitly classified as Output Data.
3. IP uploads protected Input Data to VMs utilising a secure channel established by TEE, preventing the protected Input Data from being disclosed to anyone other than the IP (at the Service Node applying the protection).

Table 1. Security requirements *(continued)*

SYS-9.3	The System shall not execute any computing functions other than the ones approved in the Computation Task Agreement.

How met:

1. The client-side IO Service verifies that each of the Computation Task Orchestrators enforces the expected CTA before uploading any Input Data. Assuming at least one honest Computing Node, the CT can not execute any unauthorised computing functions under MPC as that requires collective agreement from each Computing Node.
2. The verification is backed by trusted hardware. Convincing the IO Service otherwise assumes that the attacker has either discovered a way to inject the CT VM with malicious code, or broken the TEE remote attestation.

SYS-9.4	Any Interim Data produced in the System during the computation shall not be disclosed to any party.

How met:

1. The System does not provide an interface that would enable any Interim Data to be downloaded or queried. The System enables querying only the main data analysis algorithm's output that is explicitly classified as Output Data.
2. Each Computing Node executes a CT in a Trusted Execution Environment (see ADR-1). The TEE prevents executed code tampering, data tampering, and data extraction (including data or VM snapshotting).

SYS-9.5	The System shall not deliver any information to the Output Parties other than the final result predefined in the Computation Task Agreement.

How met:

1. The Computation Task Orchestrator classifies each Protected Data element using Slots (see Section 4.3.2.5).
2. Computation Task Orchestrator does not authorise requests to extract the contents of any Slot other than those marked as Output Data.

SYS-9.6	The final computation result shall be disclosed only to the intended stakeholder(s) identified as Output Parties in the Computation Task Agreement.

How met:

1. The CTA contains the identities (PAKs) of OPs.
2. To download Output Data, the Data Analyst authenticates themselves using the PAK.
3. The Computation Task Orchestrator authorises the request only if the Slot of the requested data specifies the PAK as its recipient.

Table 1. Security requirements *(continued)*

SYS-10.2	The System shall maintain an audit trail that logs significant events, including the signing of Computation Task Agreements, Input Data submission, Computation Task execution, and Output Data retrieval.

How met:

1. Each Computing Node has a local System Log Collector component that collects and stores logs emitted by the Computation Task Orchestrator in the CT VM. This includes state changes: retrieving Input Data, executing Algorithms, and providing Output Data.
2. Each Service Node has a local System Log Collector component that stores logs of seen messages. Operations on CT Artefacts are messages recorded by the Service Broker in every Service Node that sees it. Providing a signature thus constitutes as a message, that will eventually be recorded in the Client Nodes, the Root Service Node, and Computing Nodes.

5.2 Usability Perspective

5.2.1 Usability requirements

Table 2. Usability requirements

SYS-3.1	Preparation, configuration, and execution of a Computation Task in the System should be as simple and lightweight as possible for the Clients and should involve only minimal marginal costs.

How met:

1. The Client users are empowered by the Service Portal user interface to execute all of the mentioned actions as described in Section 4.1.1.
2. User interactions with the System are transparent: its not necessary for the Data Custodian, Data Analyst, Task Organiser, or Signatory to understand the structure of System. They only need to know the URL of the local Service Node.
3. The workflow complexity and knowledge requirement can be higher depending on the required level of protection and control. This varies between Clients and the risk level of the CT. The System accommodates both simple and scrutinous workflows. For example: preparation of a CT involves the Task Organiser approving a CTSC, which is a matter of clicking a button. If their participation involves only minimal risk however, they may choose to skip thorough inspection of the task details, inspecting the algorithm for correctness, or verifying the identities of Peers.

BUS-5.1	All operations within the System shall be as simple and lightweight as possible for the Clients.

How met:

1. As described in Section 4.2.1 the flexibility in deployment allows for the Client environment to be scalable in terms of security and their organisational structure. Cost of operations is proportional to the Client's requirements, furthermore they may evolve their part of the system gradually over time if need be. The minimal configuration of a single Service Node hosted by the Client is sufficient to use the System, and does not require specialised hardware, infrastructure dependencies, or significant technical prowess. The Client Software can be containerised to be deployed anywhere, including personal workstations.
2. The System incorporates automatic key rotation for PKCs (see Section 4.4.2.3), avoiding the need to periodically create new keys and certificates, saving on operational cost.

Table 2. Usability requirements *(continued)*

BUS-5.2	The System shall provide Clients with an up-to-date overview of active workflows and status updates for ongoing Computation Tasks.

How met:

1. The Artefact Manager component shows Users the consolidation process and deployment status of a CT through accessing the local Service Portal.
2. The Telemetry Aggregator component provides observability to instantiated CTs by relaying progress reports and error messages to the relevant Service Nodes. This enables the Clients to be informed of the results of data quality validation, the state of CT execution, or the nature of any blockers or errors.
3. Status updates that require timely actions are sent by e-mail, directing the User to the Service Portal.

SYS-3.2	Technical requirements for the Clients of the System in the role of an Input Party and/or Output Party should be minimal – without the need to install specialised hardware.

How met:

1. The System's components deployed in the Client's infrastructure assume generic hardware, e.g. x86 machines with Linux or Windows (with virtualisation support) operating systems. In this case, the System does not require any specialised hardware to function.
2. Even though Clients upload Protected Input Data to and download Protected Output Data from Computing Nodes' TEEs, such communication does not require any specialised hardware on the Client's side.
3. Remote attestation of TEEs performed by the Clients require specialised hardware only on the CP's side.

SYS-8.9	The Computation Task should support server-side data validation or data quality checks during Protected Input Data upload based on custom data quality assurance algorithms, part of the Computation Task Specification.

How met:

1. Input Data upload handles the initial validation of data quality locally, identifying data formatting and schema errors w.r.t the Input Specification contained in the CTSC (Section 4.4.1.2). This provides the Data Custodian quick feedback before invoking the upload procedure.
2. The optional server-side data quality assurance algorithm, which provides richer validation, is run asynchronously after a successful upload under MPC. Since this may be a long running computation, the Data Custodian is notified of the result by e-mail after it has finished. The Telemetry Aggregator informs Clients of any identified issues encountered during data quality validation.

Bibliography

[1] Riivo Talviste, Kert Tali, Mihkel Haav, and Hendrik Eerikson. *JOCONDE D1.1 Usage Scenarios and System Requirements*. Tech. rep. Available: https://cros.ec.europa.eu/joconde. 2025.

[2] Pille Pullonen-Raudvere, Armin Daniel Kisand, Riivo Talviste, and Kert Tali. *JOCONDE D2.1 Technology Survey and Analysis*. Tech. rep. Available: https://cros.ec.europa.eu/joconde. 2025.

[3] N. Rozanski and E. Woods. *Software Systems Architecture: Working with Stakeholders Using Viewpoints and Perspectives*. Addison-Wesley, 2012. ISBN: 9780321718334.

[4] Vincent Hu, David Ferraiolo, Richard Kuhn, Adam Schnitzer, Kenneth Sandlin, Robert Miller, and Karen Scarfone. *Guide to Attribute Based Access Control (ABAC) Definition and Considerations*. Tech. rep. NIST Special Publication (SP) 800-162, includes updates as of 08-02-2019. Gaithersburg, MD: National Institute of Standards and Technology, 2014. DOI: 10.6028/NIST.SP.800-162.

Functional View Index

Components

Interfaces

Glossary

Algorithm
Function(s) applied to the Input Data to produce Output Data, detailing the exact analysis to be executed by the CT.

Asymmetric Signing Key Pair
Key pair identifying a Service Node. Users with the Task Organiser role in a Service Node can use this key pair to sign CTSAs.

Asymmetric Authentication Key Pair
This key pair is used by a data custodian to interact with a CT. The public key is signed by the key pair of the parent and stored as a PKC

Client Software
The service node software with all the components required for usage by a client.

Client Node
Alias for the Service Node of the Client.

Computation Task Log
Logs concerning a specific CT, e.g. its state changes and any logging output produced by the data analysis algorithm itself

Computation Task Artefacts (CT Artefacts)
Any data which is part of the CTA or related to its lifecycle. This includes, for example, an explanation for rejection when the system operator rejects a CTSA, the Computation Task VM addresses, or the current execution state of the respective CT. Also refer to Section B.2.

Computation Task Specification (CTS)
Definition of a CT including (among other details) a human-readable description, the corresponding computation Algorithm, data-model, identities of all involved CPs, Users and assignment of roles.

Computation Task Agreement (CTA)
A legally enforceable agreement between the Clients; signed version of the Computation Task Specification.

Computation Task Specification Addendum (CTSA)
Data structure or document containing information supplied to a CTS regarding Peer-internal role assignment.

Computation Task Specification Core (CTSC)
Data structure or document containing the core information of a CTS, i.e. including everything but the Peer's internal role assignment information.

Computation Task (CT)
Manifestation of a CTA which is deployed across the distributed MPC infrastructure.

Computation Task Virtual Machine (CT VM)
a TEE-based virtual machine provisioned to host the functions of one CT at one CP.

Computing Node software
software package(s) deployed by the CP on their own infrastructure.

Computing Node
Alias for the Service Node of the Computing Party.

Control Plane
One of the three conceptual elements of the System, encompassing the planning, deployment, and coordination of CTs.

Data Slot (Slot)
Reference designator for data elements in a CT; subject to access control and data lifecycle rules.

Data Plane
One of the three conceptual elements of the System, encompassing any operation on Restricted Data.

Hardware Security Module (HSM)
Dedicated hardware which securely manages, stores and uses cryptographic keys. In JOCONDE their use is recommended for storing Asymmetric Signing Key Pairs.

Input Data
The pre-existing data (sets) used as input for a CT.

JOCONDE System (System)
Subject of this architecture description; an ICT solution implementing the MPSPCaaS concept, whereby ESS members and their partners could perform on-demand SPC tasks on their respective data without sharing it in intelligible form, neither with each other nor with an external Trusted Third Party.

Key Pair
One of the following, context dependent: (1) The Asymmetric Signing Key Pair which identifies a service node, or (2) the Asymmetric Authentication Key Pair of a data custodian.

Management Plane
One of the three conceptual elements of the System, encompassing System management and auditing activities.

Member's information
Information held by the SO regarding a Member that is exchanged during onboarding.

Output Data
The new data (sets) combined from the Protected Output Data.

Peer
The peers are all the clients which participate in one specific CTA.

Plane
Top-level conceptual element of the System that categorises responsibilities and functionality.

Protected Data
A representation of data that is made illegible by applying cryptographic techniques specific to the MPC technology. The representation allows the System to perform computations on the data without removing the protection.

Public Authentication Key (PAK)
The public key of the Asymmetric Authentication Key Pair. Used within the CTA to grant data upload or data download permissions to Data Custodians.

Public Key Certificate (PKC)
This certificate contains the public key of the Asymmetric Signing Key Pair of a service node as well as additional meta information to identify the service node. It is signed by the Asymmetric Signing Key Pair of the parent Service Node.

Restricted Data
: Confidential Input Data or Output Data that must be kept secret from other Members of the System and third parties due to regulatory (e.g. data protection or confidentiality requirements) or other reasons.

Role Based Access Control (RBAC)
: approach for restricting access of users (or entities) based on their defined roles.

Root Service Node
: The Service Node of the SO

Secure Multiparty Computation (MPC)
: Technique for evaluating a function with multiple peers so that the agreed party learns the output value but not each other's inputs.

Service Node
: A server which is hosting System components, offering a concrete set of functionalities to other Service Nodes and Users. Usually further classified by its main role

System Agreement
: A legally enforceable agreement between a Member and the SO regarding the membership in and use of the System

System Log
: Logs that transcend any specific CT

Trusted Execution Environment (TEE)
: An input privacy technique based on specific CPU extensions.

System stakeholders

Approver
: Individual or committee who green-lights CTSs

Auditor
: Individual appointed by the SO to run auditing activities in the System as a SA

Client
: Member who uses System facilities to perform Computation Tasks on-demand.

Computing Party (CP)
: An independent Member in the system who owns, operates or otherwise provides a Computing Node in order to execute Computation Tasks.

Data Analyst
: Individual who processes Output Data

Data Custodian
: Individual who carries out the preparation and submission of Input Data

General Manager
: An individual running day-to-day operations on the Management Plane

Input Party (IP)
: Member who provides Input Data for a Computation Task.

IT Administrator
: Individual of some organisation (e.g. a Member's or SO's) who tends to technical components and subsystems, carrying out their deployment and maintenance activities

Member
: Legal persons or other entities with autonomous information systems who have been accepted by the SO to interface with the System

Output Party (OP)
: Members who receive Output Secret Shares from a specific Computation Task.

Signatory
: Individual with legal authorisation to sign documents on behalf of their organisation

System Auditor (SA)
: An authorised entity (independent from the SO) who provides auditing services to verify the correctness of the operation of the System by, for example, detecting errors, attacks or attempts to deviate from the System workflow.

System Operator (SO)
: Eurostat (European Commission) within the capacity to manage membership and coordinate communication between Members in the System.

Task Organiser
: Every Peer has a set of task organisers who participate in the creation of the CTA. There can be more than one task organiser per Peer to allow to cover in the case of absence. A task organiser is often an Data Analyst as well.

User
: Natural person authorised by a Client to use the system

Appendix A Architectural Decision Records

The architectural decisions made here are documented using Architectural Decision Records (ADR) as described by the ADR GitHub organisation[1]. Michael Nygard's decision record template was used for this purpose[2].

ADR-1 Combination of TEE and MPC

Status Accepted.

Context With the security guarantees of MPC protocols considered, compromised Computing Nodes can still pose a threat. For example, by tampering Computing Node software. In this scenario, tampered Computing Node software could enable an attacker to carry out a CT that deviates from the CTA or MPC protocol to violate confidentiality of Restricted Data. By ensuring integrity of strategic elements of the computing environment, this threat would be mitigated.

If the adversary were to gain access to all Computing Nodes, they can still walk away with the Restricted Data. Trusted hardware can offer the additional layer of security by providing confidentiality of Protected Data (i.e. individual secret shares) in transit, at rest, and in use.

Decision The System is to utilise a virtual-machine-based TEE with hardware-based encryption to encase the (1) Control Plane components responsible for executing a CT and (2) the MPC Engine executing the MPC protocol (MPC-over-TEE model [2]). One CT constitutes one VM instance per Computing Node, that remain running for the entire lifecycle of the CT. VMs are instantiated based on a public and auditable disk image, containing (1), (2), alongside the operating system, kernel, etc. Each image has a measurement that is known to (and verifiable by) anyone – it is subsequently verified, by a Data Custodian or other Computing Nodes, that the VM is based on that image by running a remote attestation procedure. The remote attestation is combined with a protocol to establish a secure channel for the transfer of Protected Data and MPC protocol messages. Data at rest must be encrypted with an ephemeral key that is bound to the CT VM. To mitigate the risk of TEE vulnerabilities, the System is to use TEEs from different vendors.

Consequences

1. Computing Node software is protected against tampering. Remote attestation provides the integrity guarantee backed by hardware.
2. Secure channels provide hardware security to data in transit.
3. Protected Data in use is additionally protected by memory encryption.
4. Any data persisted by the CT is left illegible once the CT VM terminates.
5. Software components (e.g. adapters) that enable usage of different TEEs (e.g. VM provisioning) are required.

[1] https://adr.github.io
[2] https://cognitect.com/blog/2011/11/15/documenting-architecture-decisions

6. Certain MPC protocols enjoy hardened security guarantees.

ADR-2 Decentralise services

Status Accepted.

Context To simplify Users' interactions with the System, a front-end component is needed that would be accessible over the network. In D1.1, this component is referred to as the *Client Portal*. This component furthermore interacts with other components and these interactions also have to be trusted by the Clients. As the System's emphasis is both on the *no single point of trust* aspect and usability, it is important to come up with a solution that would not negatively affect usability of the System while also not introducing any single point of trust.

Decision Having a centralised and System-Operator-managed Client Portal with services to-be-interacted would increase the number of required measures to ensure the System can be trusted by the Clients, reducing the usability and complicating the development of the System. Thus, the decision is to decentralise the System's services and deploy them to the Clients' premises. The subset of such services is referred as the *Client Software*.

Consequences

1. Simplifies trust establishment: Clients deploy the services on-premises and can trust them without overhead (even though source code review before deployment is required).
2. Reduces the number of measures necessary for trust establishment.
3. Complicates development: communication between multiple nodes is needed to make the System work.
4. More responsibilities are delegated to a Client's IT personnel (can be partially mitigated through the provision of deployment scripts and other tools).

ADR-3 Service Portal Instead of Client Portal

Status Accepted.

Context To simplify Clients' interaction with the System, a front-end component accessible over the network is needed. A similar component is also needed for the SO for interactions with the System that are not captured by the Client Software. However, there are some common functionalities expected by both Clients and the SO, such as access control, telemetry services, and CTS signing. Yet only the SO must be able to submit the CTA or manage access to the System whereas only Clients need to apply protection to Input Data.

Decision One option is to create two separate sets of software for Clients and the SO. Even though this would mean explicit specification of which Portal components are to be deployed and where, it would make the System development more complex. Thus, the decision is to continue with a single set of software referred as the *Service Portal* that would be utilised by both the Clients and the SO.

Consequences

1. Simplifies the development of the System: no need to maintain three separate programming modules (e.g. for common, Client-specific, and System Operator-specific functionalities) but just one. Potentially also reduces the amount of boilerplate code.
2. Requires Role Based Access Control (RBAC): Clients must not be able to use System Operator-specific functionalities and vice versa.
3. Requires a decision explaining how to distinguish between the SO's and Clients' Service Nodes.

ADR-4 Tree topology, propagation via Service Broker

Status Accepted.

Context This deliverable introduces multiple intra-Client stakeholders such as *Data Custodian*, *Data Analyst*, and *Signatory*. This is because the System should be usable by organisations of different sizes, regardless of their internal structure. For example, in an organisation with a complex internal structure, the Data Custodian (who uploads Input Data and downloads Output Data) and Signatory (who signs the CTS) may be different individuals. The differences in roles mean such individuals are likely to have different permissions for System usage. Traditionally, this would be addressed by RBAC, which would not, however, solve all problems relevant in the context of the System.

1. Multiple Data Custodians may exist in a single organisation, each of them handling and in charge of different data.
2. In a corporate setting, it is not always reasonable to assume that the Data Custodian can reveal Restricted Data (subject to a Computation Task) to the wider infrastructure of the organisation in plain.

Instead, Data Custodians should be able to apply protection to Input Data locally and upload it directly to Computing Nodes.

Decision There are multiple ways to avoid intra-organisational transfer of unprotected Input Data.

1. (a) Implement a browser-based program with data protection and CT integrity verification components integrated into Javascript / WebAssembly.

   (b) Establish processes for cryptographic key management and protection.
2. (a) Implement a lightweight version of the Service Node software (which already contains the required functionality) with data protection and CT integrity verification components but without functionality not needed by a Data Custodian.

   (b) Establish processes for cryptographic key management and protection.
3. (a) Enable Data Custodians to use a self-hosted, on-premises Service Node that they trust.

   (b) Reuse the same processes for cryptographic key management and protection established for regular Service Node without additional specialisation.

The first option is technically complex, the second is simpler but both require overhead of specialized processes establishment and maintenance of multiple programming modules. The third option does not require such overhead at all, and we will proceed with this option.

On the other hand, connecting each and every intra-Client Service Node directly to the SO's Service Node would require execution of onboarding and certification processes for each instance separately. To avoid such overhead, the decision has been made to us a tree topology for the Service Nodes. In this tree, the root would be the SO's Service Node. Its direct children would be the 'central' Service Node within the IPs' and OPs' organisations, as well as the CPs' Service Nodes. Each such child could then have its own children (which would apply to the Data Custodian's scenario). In practice, this tree could grow even deeper depending on organisational structure (probably with multiple sub-departments).

This topology requires a mechanism to propagate non-private data between the tree's nodes called the *Service Broker*. One example of propagation need is CTS synchronisation between all involved parties. This scenario is illustrated by Figure 39. As shown in the figure, each Service Node has a Service Broker but only the involved parties participate in CTS propagation (depending on the parties specified in CTS). The respective CTA would be propagated to the same parties, as well as the Computing Nodes, but this would take place downstream from the SO's node.

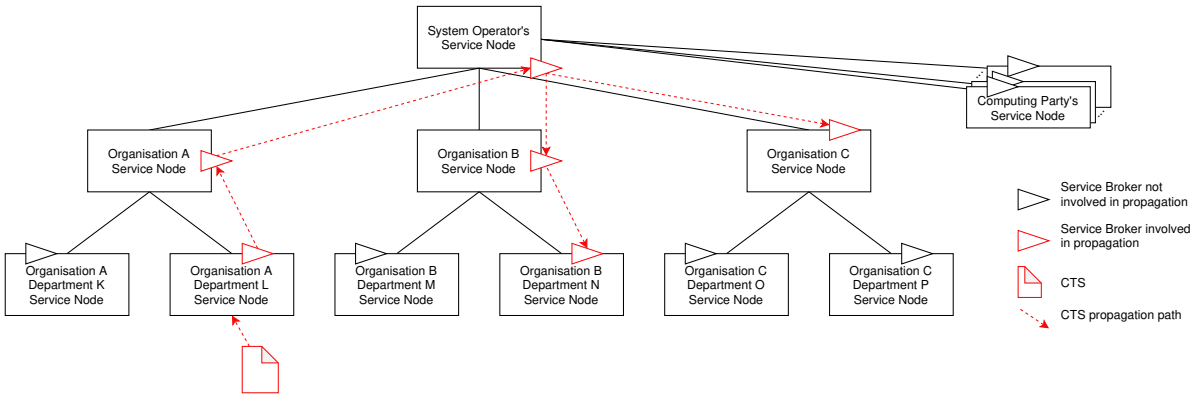


Figure 39. Example of CTS propagation.

Consequences

1. No need to trust intra-organisational nor the SO's infrastructure.
2. The System can be used in organisations regardless of their internal structure.
3. The SO does not need to execute onboarding and certification processes for internal Service Nodes.
4. Possibility to utilise a fine-grained RBAC.
5. No need to maintain specialised programming modules and processes.
6. Needs a mechanism enabling the Service Broker to do routing.
7. Increases the importance of machine-readable CTSs and CTAs.
8. Delegation of infrastructure to the Data Custodian implies lowering the operational burden of deploying the Service Node.

ADR-5 Certificate chain per node

Status Accepted.

Context Introduction of tree topology (see ADR-004) for Service Nodes raised several questions.

1. How to route a CTS/CTA in multicast (and not broadcast) mode when the recipients must be the nodes involved in a particular CT consolidation?
2. How to establish secure channels between interacting Service Nodes?

All these questions can be addressed within a single ADR.

Decision The decision is to utilise X.509 certificate chains issued per each Service Node. This approach requires having a hierarchical system of certificate authorities (CA) that issue certificates chains. The hierarchy corresponds to the tree topology described in ADR-4. In other words, each Service Node is to be a CA with the root CA of SO. In contrast to the TLS/SSL protocol, our public key infrastructure (PKI) is not intended to provide trust as is, since this would introduce a single point of trust that must be avoided. The System distributes certificate chains of other Service Nodes but trust can be established only when combined with out-of-band measures. Trust establishment toward a particular Service Node involves at least following:

1. Reception of a certificate chain of the Service Node from within the System.
2. Generation of a fingerprint from the certificate chain.
3. Retrieving the fingerprint of the Service Node's certificate chain, communicating with the Organiser who is local to the *target* Service Node via channels outside the System.
4. Comparison of the retrieved fingerprint in step 2 with the fingerprint in step 3. They have to be the same to trust the Service Node.

Each Service Node thus has to maintain its trusted view of other Service Nodes (their certificate chains) this particular Service Node interacts with. Trust towards a peer is built by having previously participated in computations with said Service Node, as this implies that the procedure has been completed beforehand, and by making subsequent or recurring collaboration easier and more secure. For this purpose, each Service Node must have its local Node Identity Manager. This component is primarily responsible for storage and propagation of certificate chains. But the component also enables permissions management, authentication, and authorisation based on the certificate chains. With a trusted view of other Service Nodes, their certificate chains can be associated with a set of permissions.

Consequences

1. A feature is needed that would discourage the User from trusting certificate chains without applying out-of-band measures (could be a notification at least), becoming more important than previously.
2. Creates a mechanism for routing CTS/CTA.
3. Certificate chains fulfil the prerequisite for secure channel establishment.
4. Certificate chains fulfil the prerequisite for implementing role based access control for Service Nodes.

Appendix B Code Examples

B.1 Illustration of the CTA Data Type

```
struct CTA {
  cts: CTS,
  qe_signatures: Vec<SignatorySignature>,
  node_signatures: Vec<NodeSignature>,
}

struct CTS {
  ctsa: Vec<CTSA>,
  ctsc: CTSC,
}

struct CTSA {
  party_spec_index: usize,
  signatories: Vec<Signatory>,
  data_custodians: Vec<DataCustodian>,
  sha256_hash_of_ctsc: [u8; 32],
}

struct DataCustodian {
  public_authentication_key: Vec<u8>,
  input_specs: Vec<String>,
  output_specs: Vec<String>,
}

struct Signatory {
  /// SO should already have this document within its member information
   database.
  signatory_identification_document: File,
}

struct SignatorySignature {
  ctsa_index: usize,
  signatory_index: usize,
  signature: SignatureFile,
}

struct NodeSignature {
  party_spec_index: usize,
  // Type to be specified
  signature: Vec<u8>,
}

struct CTSC {
```

```
    human_readable_title: String,
    human_readable_description: String,
    attachments: Vec<Attachment>,
    temporal_spec: TemporalSpec,
    computation_spec: ComputationSpec,
    peer_service_node: Vec<CertificateChain>,
    computing_party: Vec<CertificateChain>,
}

struct Attachment {
  human_readable_title: String,
  human_readable_description: String,
  content: File,
}

enum SignatureFile {
  // Yet To Be Specified, just examples.
  AsicE(Vec<u8>),
  EcdsaSignature(Vec<u8>),
}

enum File {
  // Yet To Be Specified, just examples.
  PDF(Vec<u8>),
}

struct TemporalSpec {
  approval_deadline: chrono::DateTime<Utc>,
  input_deadline: chrono::DateTime<Utc>,
  execution_deadline: chrono::DateTime<Utc>,
  output_deadline: chrono::DateTime<Utc>,
}

struct CertificateChain {
  // Specific format (concatenated X.509 PEMs, or PKCS#7) to be determined
    .
  chain: Vec<u8>,
}

struct ComputationSpec {
  mpc_engine: String,
  /// The type is just a placeholder and should be specified.
  tee_measurements: Vec<u8>,
  main_algorithm: Algorithm,
  input_specs: Vec<InputSpec>,
  output_specs: Vec<OutputSpec>,
}

struct InputSpec {
```

```
  key: String,
  party_spec_index: usize,
  schema: Vec<Column>,
  data_quality_algorithm: Option<Algorithm>,
}

struct OutputSpec {
  key: String,
  party_spec_indices: Vec<usize>,
  schema: Vec<Column>,
}

struct Column {
  name: String,
  data_type: DataType,
}

struct Algorithm {
  // To be specified.
  source_code: String,
}

enum DataType {
  // Yet To Be Specified, just examples.
  U8, U16, U32, U64, U128,
  I8, I16, I32, I64, I128,
  F8, F16, F32, F64, F128,
  Bytes,
}
```

B.2 Illustration of the Database Schema

```
create table child_nodes (
    id uuid primary key,
    active_certificate bytea null,
    -- Packed into one BLOB, the application layer can unpack it.
    -- No fast lookups required in our architecture.
    old_certificates bytea null,
);

-- Just one parent node is configured.
create table parent_node (
    active_certificate bytea null,
    old_certificates bytea null,
);

create type cta_status as enum (
    'emerging',
    -- As soon as it is approved, it will be sent out to the
    -- computing parties for instantiation.
    'instantiated',
    'cancelled',
```

```
    'finished'
);
create table cta(
    id uuid not null primary key,
    -- Packed into one BLOB, the application layer can unpack it.
    ctsc_data bytea not null,
    status cta_status not null,
    status_human_readable_context text,
    -- Shall be updated whenever this row or a corresponding row in the
    -- "ctsa" table changes. This value shall be displayed in the user
    -- interface. Also, it can be used by child service nodes, which
    -- request an overview of all their CTA, to exclude rows which have not
    -- changed for a longer time. To avoid ambiguities with time comparison,
    -- this shall be coarse, and clients cross check against the
    -- "cta_update_counter" locally.
    last_changed timestamptz not null,
    -- Shall be increased whenever this row or a corresponding row in the
    -- "ctsa" table changes. This value is sent to child service nodes
    -- such that they can determine whether they have stale information.
    cta_update_counter int not null DEFAULT 0,
    -- Used by data custodians, data analysts and other TEE VMs of the same
    -- computation task to find the VM. Sent by each computing party to the
    -- root service node, accumulated into this field. `jsonb` allows inline
    -- appending. Just one blob is required as usually the user of this field
    -- wants to know all of the present values, so no need to normalise.
    tee_vm_addresses jsonb null,
    INDEX (id, status),
);

create type ctsa_status as enum (
    'fresh',
    -- The SO may reject the addendum. In this case the peer
    -- needs to apply fixes.
    'rejected',
    'approved',
);
create table ctsa(
    id uuid not null primary key,
    cta_id uuid not null references cta(id),
    -- Packed into one BLOB, the application layer can unpack it.
    ctsa_data bytea not null,
    status ctsa_status not null,
    -- If the addendum status is 'rejected', this must contain a
    -- text of what to fix.
    status_human_readable_context text,
    -- Shall be updated whenever this row changes. This value shall be
    -- displayed in the user interface.
    last_changed timestamptz not null,
    INDEX (cta_id, status),
);

-- A table to allow for quick lookup of CTAs by the pubic key.
-- This is used by (1) peers and (2) data custodians/analysts to lookup CTA data.
-- Public keys of service nodes are added when a new CTSC is added. Public keys
-- of data custodians and analysts are added when a CTSA is added, and deleted
-- when the CTSA is replaced.
create type public_key_type as enum ('service_node', 'dc_or_a');
create table public_key_2_cta_lookup(
    -- This is the raw public key and not the PKC, even for peers.
    public_key bytea not null,
```

```
    cta_id uuid not null references cta(id),
    public_key_type public_key_type not null,
    -- For (1) service_node this is the expiration date of the PKC, and for (2)
    -- dc_or_a it is the input deadline (data custodian) or output deadline
    -- (analyst) specified in the CTSC.
    expiration_date timestamptz not null,
    UNIQUE (cta_id, public_key),
    INDEX (expiration_date),
);

-- Used to:
-- (1) Check that to-be-inserted CTSC does not contain revoked keys.
-- (2) Member nodes cannot connect with revoked keys.
-- (3) Member nodes cannot register a new key which has been revoked in the past.
create table revoked_keys(
    public_key bytea not null primary key,
)
```