UNIVERSITY OF TARTU
Institute of Computer Science
Software Engineering Curriculum

Anton Zakatov

Architecture of Secure Multi-Party Computation as a Service with No Single Point of Trust

Master's Thesis (30 ECTS)

Supervisors:
Kert Tali, MSc
Raimundas Matulevičius, PhD

Tartu 2025

Architecture of Secure Multi-Party Computation as a Service with No Single Point of Trust

Abstract:

Information privacy is receiving significant attention in both legislative and public discourse. Increasing awareness of the risks inherent in cross-organisational data analysis has led to limitations on traditional statistical approaches. Although secure multi-party computation (MPC) offers a promising solution to risk mitigation, existing MPC technologies remain too resource-intensive for widespread practical adoption. The JOCONDE project seeks to overcome this problem with a novel MPC as a Service system. This thesis contributes to the system design by addressing its challenge – to circumvent a single point of trust. No entity should be able to compromise sensitive data – the foundational security guarantee of MPC, which must be extended to the new system. A threat model has been developed to reach the goal, serving as the basis for formulating appropriate countermeasures and architectural decisions. Experts validated the results and confirmed the completeness of the threat model along with the sufficiency and feasibility of single-point-of-trust mitigations.

Keywords: secure multi-party computation, as a service, software architecture, single point of trust

CERCS: T120 Systems engineering, computer technology

Kriitilise usalduspunktita arhitektuur turvalise pilvühisarvutuse süsteemile

Lühikokkuvõte:

Teabe privaatsus on laialdast tähelepanu pälvinud nii seadusandluses kui üldsuse seas. Teadvustatakse muuhulgas riske, mis kaasnevad andmete töötlemisega organisatsioonide üleselt, piirates levinud statistika tootmise metoodikaid. Turvalise ühisarvutuse (MPC) tehnoloogiad pakuvad probleemile lahendust, kuid praktikas osutub nende rakendamine liiga ressursimahukaks. JOCONDE projekti algatusel otsitakse ressursiprobleemile lahendust uudse turvalise pilvühisarvutuse süsteemiga. Käesolev lõputöö sisaldab osa süsteemi disainist, mis keskendub kriitilise usalduspunkti (*single point of trust*) vältimisele. Aluseks olev turvalise ühisarvutuse turvagarantii, mis tagab, et ükski osapool pole võimeline tundlikke andmeid kompromiteerima, peab laienema ka uuele süsteemile. Selle nimel loodi ohumudel, millest lähtudes disainiti meetmed ning koostati komplekt arhitektuurilisi otsuseid. Saadud tulemusi valideeriti ekspertintervjuudes – kinnitati ohumudeli täielikkus, lahenduse piisavus kriitilise usalduspunkti vastu ning otsuste äriline teostatavus.

Võtmesõnad: turvaline ühisarvutus, pilvelttarkvara, tarkvaraarhitektuur, kriitiline usalduspunkt

CERCS: T120 Süsteemitehnoloogia, arvutitehnoloogia

Contents

1. Introduction

Nowadays, there is more and more emphasis on data privacy. In 2018, the General Data Protection Regulation, or GDPR, was enacted in the European Union. This law protects the privacy of natural persons and regulates how personal data can be processed [1]. There are different personal data protection laws in non-EU countries as well. At the same time, personal data is irreplaceable for informed decision-making, the true power of which lies in the statistically derived insight that personal data can provide.

Statistical products often require input from multiple sources from different organisations. The known approach to cross-organisational statistics production is to share data ahead of processing [2]. Sharing introduces additional risks to data privacy and confidentiality, challenging the prospect of the task. Limitations may apply from legislation or insufficient trust between data holders, including companies and organisations, data processors (e.g., statistical authorities), and data subjects such as the general public. These concerns are not new and have been addressed by researchers of privacy-enhancing technologies, who aim to develop data processing methods that preserve confidentiality.

One solution for privacy-preserving data analysis is carrying out a secure multi-party computation (MPC). MPC enables multiple data holders to jointly compute a specified function without revealing the inputs to anyone else. The only information learned is the output of the function [3].

Even though different sophisticated MPC solutions are available for real-life scenarios, their usage is complex and resource-intensive [2]. Thus, in 2023, the statistical office of the European Union, or Eurostat, put out a public tender for a specification of an MPC as a Service system (hereinafter *System*) [4, 5]. The System aims to enable the production of statistics in scenarios where data sharing would otherwise be necessary [5]. It ensures the process is simple, lightweight, and low-cost for organisations participating in joint computations.

It follows that the System must not have any single point of trust (SPoT) that would enable the misuse or disclosure of any sensitive data of its Users. In other words, this means that the System must, by design, eliminate the ability to compromise the confidentiality of a Client's sensitive data by one single entity (including the service providers) [4, 5].

As a result of the tender, Eurostat in collaboration with the Estonian company Cybernetica AS[1] carries out a project JOCONDE (**J**oint **O**n-demand **CO**mputation with **N**o **D**ata **E**xchange)[2] [6]. This thesis is closely aligned with this project. While the project delivers a comprehensive analysis and specification for the System, the thesis focuses on a single architecturally significant security requirement.

The goal of this study is to formulate a set of architectural decisions to circumvent single point of trust in the System. Decisions presented in this thesis capture important design elements of a larger system, and are thus influenced by context: project requirements, scope, and other design activities taking place in parallel. The major contribution of this thesis is the solution and rationale that accompanies decisions – justifying how and why the System has to be built in a certain way.

The following research questions are intended to help reach this goal. The main research question (MRQ) is: "How to achieve no single point of trust for the System?". The MRQ consists of multiple minor research questions (RQ).

RQ1. How can architectural decisions be formulated to design the System with no single point of trust?

RQ2. What architectural decisions enable the design of the System with no single point of trust?

RQ3. Do the formulated architectural decisions achieve the System with no single point of trust?

This thesis addresses the unsolved problem before and examines how to design a privacy-preserving system without making it just a complex trusted third party. The contribution of this study lies in the identification and justification of architectural decisions that ensure the System can operate without any single point of trust. To solve this problem, we 1) construct a threat model for a baseline architecture, 2) formulate architectural decisions based on the identified threats, 3) revisit the threat model taking into account the specifics of the System and the decisions, 4) and validate our results through expert interviews. As this study aims to design

[1]Webpage of Cybernetica AS: https://cyber.ee/
[2]Webpage of the JOCONDE project: https://cros.ec.europa.eu/joconde

an innovative artefact, we follow a design science research paradigm in interpretation by Hevner et al. [7].

In Chapter 2, we provide background on MPC and MPC as a Service. We then examine the concept of single point of trust and outline the approaches of architecture design and threat modelling. Finally, we present an overview of the System. Chapter 3 describes the methodological steps used to identify threats, formulate architectural decisions, and validate the results. Chapter 4 presents the results, detailing the developed threat model and architectural decisions. Chapter 5 outlines the evaluation of the results. Chapter 6 concludes the thesis by answering the main research question, discussing limitations of the study and proposing directions for future research.

General Terms

architectural decision

Justified design choice that addresses a functional or non-functional requirement that is architecturally significant [8].

architecturally significant requirement

Requirement that has a measurable effect on the architecture and quality of a software and/or hardware system [8].

architecture

Fundamental concepts or properties of an entity in its environment and governing principles for the realisation and evolution of this entity and its related life cycle processes [9].

secure multi-party computation

Technique for evaluating a function with multiple Peers so that the agreed party learns the output value but not each other's inputs.

trusted execution environment

Input privacy technique based on specific CPU extensions.

JOCONDE Glossary

This glossary compiles key terms used throughout the thesis, as developed in the context of the JOCONDE project. The definitions are drawn from the report “JOCONDE D4.1. System Specification and Architecture” [10].

Algorithm

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

Client

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

Computation Task

Manifestation of a Computation Task Agreement which is deployed across the distributed MPC infrastructure.

Computation Task Agreement

Legally enforceable agreement between the Clients; signed version of the Computation Task Specification.

Computation Task Specification

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

Computing Node

Alias for the Service Node of the Computing Party.

Computing Node software

Software package(s) deployed by the Computing Party on their own infrastructure.

Computing Party

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.

Input Data

Pre-existing data (sets) used as input for a Computation Task.

Input Party

Member who provides Input Data for a Computation Task.

Member

Legal persons or other entities with autonomous information systems who have been accepted by the System Operator to interface with the System.

Output Data

New data (sets) combined from the Protected Output Data.

Output Party

Member who receives Output Data from a Computation Task.

Peer

Client who participates in one specific Computation Task Agreement. The Peers are all the Clients which participate in one specific Computation Task Agreement.

Protected Data

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.

Protected Input Data

Protected Data derived from Input Data.

Protected Output Data

Protected Data that is output of a Computation Task.

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.

Service Node

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

Service Node software

Software package(s) deployed by the Client on their own infrastructure.

Signatory

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

System

ICT solution implementing the MPC as a service concept, whereby European Statistical System members and their partners could perform on-demand secure multi-party computation tasks on their respective data without sharing it in an intelligible form, neither with each other nor with an external trusted third party.

System Auditor

Authorised entity (independent from the System Operator) 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

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

Task Organiser

Individual who participates in the creation of the Computation Task Agreement.

User

Natural person authorised by a Client to use the System.

Acronyms

ADR

architectural decision record

MPC

secure multi-party computation

SPoT

single point of trust

TEE

trusted execution environment

TTP

trusted third party

2. Background

This section addresses the research question RQ1: "How can architectural decisions be formulated to design the System with no single point of trust?". This research question concerns the approach to formulating architectural decisions. In particular, it seeks to

1. identify the properties of MPC that must be accounted for,
2. explain how MPC as a Service differs from existing MPC systems,
3. clarify the meaning of single point of trust in MPC,
4. outline specifics of the JOCONDE System,
5. identify suitable methods for threat elicitation,
6. outline approaches for documenting architectural decisions.

To support this, we first provide background information on MPC and MPC as a Service. We then elaborate on the concept of a single point of trust within the context of MPC. Following this, we describe the key concepts that guided the architecture design of the System, and present an overview of its architectural structure. These elements collectively form the foundation for conducting threat modelling and proposing measures to design the System without a single point of trust.

2.1 Traditional Statistics Production

The traditional approach of statistics gathering assumes that all input data is to be processed by a single organisation (e.g., a statistical authority) [2]. However, as highlighted in the United Nations Economic Commission for Europe Input Privacy Preservation (UNECE IPP) Project [2], collecting input data from various organisations, companies, or individuals is problematic. Integrating data from multiple sources typically requires direct data exchange between parties or indirect sharing through a trusted third party. In most cases, parties enter into agreements defining data usage terms. Nevertheless, adherence to these agreements relies solely on trust, as there are no technical safeguards to enforce compliance. This lack of enforcement mechanisms increases the risk of data breaches [2]. While techniques such as data masking can reduce these risks to some extent, they do not provide strong privacy guarantees. Consequently, there is a need for a fundamentally new approach that eliminates the potential for privacy violations. One such alternative is the use of secure multi-party computation.

2.2 Secure Multi-Party Computation

The core idea behind secure multi-party computation[3] is to analyse joint confidential input from multiple sources while preserving its privacy. MPC is a broad term whose meaning can vary depending on the context. It spans multiple levels of abstraction, from low-level cryptographic protocols to high-level frameworks. This subsection defines the existing abstraction layers within MPC.

2.2.1 Theoretical Foundations

The theory of MPC originated in the late 1970s. In 1982, Yao introduced the millionaires' problem, where two millionaires want to know who is richer without revealing the number of millions they have [11]. Three years earlier, Shamir proposed an idea of secret sharing, where a secret is split into multiple pieces, and the secret can only be reconstructed by combining all of these fragments [12]. Since then, numerous theoretical studies have been conducted to enhance MPC protocols (e.g., make them more efficient) or invent new ones.

Nevertheless, the core idea of MPC is to preserve privacy by design when processing sensitive data. More formally, MPC allows n parties to compute output $f(x_1, ..., x_n) = (y_1, ..., y_n)$ based on their inputs $x_1, ..., x_n$, while each party i knows x_i and learns not more than y_i [13].

Data processing during MPC involves different types of parties. Input Parties provide the inputs for the computation, Computing Parties carry on the computation, and Output Parties receive the output of the computation. These three sets of parties can be the same, but can also be different. Our study primarily focuses on a model in which the Computing Parties are distinct from the Input Parties. This approach is referred to as the outsourcing model, where Input Parties supply their data to a predetermined set of Computing Parties for computation [14].

There are two types of MPC: two-party (involves two Computing Parties) and multi-party (more than two Computing Parties are involved). The main cryptographic approaches used are garbled circuits, homomorphic encryption and secret sharing [3]. All these types and approaches are actively used in different MPC technologies [14, 15].

---

[3] Secure multi-party computation is also known as a *multi-party computation*. In the following part of this thesis, the terms mentioned are to be interpreted as equivalents unless otherwise stated.

Security is a fundamental aspect of any MPC protocol. As noted by Lindell [16], an MPC protocol is considered secure if it satisfies the corresponding properties. Various sets of properties define MPC security, and the same author outlines a representative, though non-exhaustive, set of these properties. This set is necessarily incomplete, as the precise requirements often depend on the specific application context. The commonly identified properties include the following:

1. **privacy** – none of the parties should be able to learn more than the predefined output of the computation;
2. **correctness** – each Output Party must receive the correct output of the computation;
3. **independence of inputs** – each Input Party must choose their input independently, even if the input is incorrect;
4. **guaranteed output delivery** – no party should be able to prevent any Output Party from receiving its output;
5. **fairness** – either all Output Parties receive the output, or none do [16].

Lindell stresses that these properties must be satisfied by any secure protocol. However, some, such as fairness or guaranteed output delivery, may be excluded in practice due to their limited feasibility in certain scenarios. The theoretical foundations of MPC establish essential security guarantees that must be preserved across higher levels of abstraction. Any such abstraction should uphold these guarantees to ensure the protocol remains secure.

2.2.2 Protocol Practical Implementations

It took some time before MPC found practical application. The first real-world use of MPC occurred in 2009 during the Danish sugar beet auction, where it was used to protect the privacy of farmers' bids [17]. A more recent (2023) example is a proof of concept proposed by Meta and Mozilla for privacy-preserving digital advertising without third-party cookies [18]. Both cases demonstrated the potential of applying MPC – and privacy-enhancing technologies more broadly – in real-world scenarios.

Despite this potential, there are notable challenges in protocol implementation. First, some protocols are computationally expensive or impose considerable overhead, complicating their practical use [3]. This problem lies outside the scope of our research. Second, prior to the introduction of higher levels of abstraction, such as MPC frameworks, deploying MPC in practice required designing both the protocol and the system architecture specifically for each

use case. Such ad hoc solutions are unscalable and demand substantial resources and expertise in relevant domains to build and maintain [2].

2.2.3 MPC Frameworks

The intermediate level of abstraction for MPC technologies is a framework. These are reusable tools that remove the need for ad hoc protocol implementations. Nowadays, several frameworks have been successfully used in practical scenarios. Examples include Roseman Labs [19], XOR [20], and Sharemind MPC [21].

MPC frameworks often bundle ready-made protocol implementations, domain-specific languages or application programming interfaces to use the protocols in high-level computation, and client and server side components [15]. They provide the building blocks for complex MPC applications.

Nevertheless, this level of abstraction does not solve all issues. First, using an MPC framework to build a system does not inherently guarantee its security [22]. A thorough understanding of the application domain remains essential even when using MPC. Second, frameworks still require ad hoc deployment. Deploying an MPC framework may involve multiple steps: selecting Computing Parties; reviewing the source code of the framework; delivering binaries to Computing Parties; configuring and activating the software on computing nodes; setting up firewalls, port forwarding, sandboxing, and virtual private networks; provisioning backup systems; sharing public keys of the computing nodes; configuring a proxy server between end-users and Computing Parties; developing the Computation Task; installing client applications for Input Data upload; documenting the data flow, infrastructure, and configuration of the MPC technology; training personnel; defining user roles, responsibilities, and access policies; signing agreements with Computing Parties regarding handling of (illegible) data (e.g., secret shares); and entering additional contracts as needed [17, 21, 23, 24]. Using MPC frameworks has been described as complex and resource-intensive [2]. Many of these processes could be simplified by introducing a new abstraction layer above the existing ones.

Every MPC framework must satisfy a core set of requirements to perform a Computation Task. Any additional abstraction layer built on top of the framework abstraction must also meet these requirements. We may conceptualise an abstract MPC framework that meets only these essential requirements, without offering additional functionalities. We refer to it as a Minimum Viable MPC Framework.

To define such a Minimum Viable MPC Framework, we draw on the following sources: documentation of Sharemind MPC [25], documentation of Roseman Labs [26], JOCONDE analysis documents [14, 27], an article on the Partisia Blockchain infrastructure [28]. Based on these sources, we identify the following requirements as essential for the operation of any MPC framework.

1. Peers must reach agreement on the details of a Computation Task, either manually or through the framework's mechanisms.
2. Input Parties must be able to protect their Input Data.
3. Input Parties must submit their Protected Input Data prior to task execution.
4. Protected Data must be transmitted via secure communication channels to and from the Computing Parties.
5. Computation Tasks must be executed by multiple Computing Parties (typically three), using pre-installed software capable of secure joint computation. This software may run on physical or virtual machines.
6. Computing Parties must store and process Restricted Data associated with a specific Computation Task.
7. Computation Tasks must reveal no information beyond the agreed output, which is accessible only to the designated Output Parties.
8. Output Parties must retrieve their Protected Output Data upon successful completion of the Computation Task.
9. Output Parties must be able to remove the protection from their Protected Output Data.

This abstract framework forms a basis for subsequent threat modelling. Although the list may be seen as incomplete or overly inclusive, it provides a practical baseline. It does not replace the need for a threat analysis tailored to a specific implementation, but offers a helpful starting point for such assessments.

2.3 MPC as a Service

Concerns about MPC frameworks have been addressed in the UNECE IPP Project [2]. The central goal is to make secure multi-party computations user-friendly and user-affordable by removing the need for ad hoc deployment and overhead processes. This goal can be achieved

by adopting the *as a service* model – an idea of offering technologies, tools, or products on a subscription basis. Examples include *Infrastructure as a Service* (IaaS) and *Software as a Service* (SaaS). Similarly, offering MPC technology as a service is conceivable.

MPC as a Service (MPCaaS) is a layer on top of an MPC framework abstraction and enables on-demand multi-party computations. The advantage of this approach is that Users (e.g., statistical institutions) do not need to maintain the MPC-related infrastructure or most of the deployment processes associated with MPC frameworks. Instead, the System and its shared infrastructure are managed by dedicated personnel with expertise in this domain. The System would leave Users responsible only for specifying and deploying a common Computation Task [5]. Moreover, the System provides flexibility for Clients in selecting appropriate MPC technologies and configuring computations on a case-by-case basis. However, to the best of our knowledge, there are no providers of multi-party computation as a service at this point[4].

Developing the System requires addressing multiple technical and legislative challenges. During the UNECE IPP Project, the work group surveyed experts and prospective Users of the System [2]. Some identified challenges include efficiency, usability, public trust, legislative uncertainty regarding MPC-based data processing, defining stakeholder responsibilities and the business model of the System. Particular emphasis is placed on designing the System with no single point of trust. In other words, no entity should possess the technical ability to compromise the confidentiality of a Client's sensitive data [4, 5]. An *entity* refers to any actor, whether an individual, organisation, or system component, that performs or is capable of performing specific functions related to the System. For example, entities include Input Parties, Output Parties, Computing Parties, the System Operator, software and hardware vendors, or any other actor with access to the infrastructure, data, management or control. Conventional information systems often have a single point of trust in the form of the service provider or software vendor. Nonetheless, no unsolvable issues have been identified that would prove the infeasibility of the System [2].

[4]It must be acknowledged that there are many advantaged MPC frameworks with some similarities to MPCaaS. While these do not correspond to MPCaaS in the way we interpret it, we give an overview of such frameworks in Section 2.9.

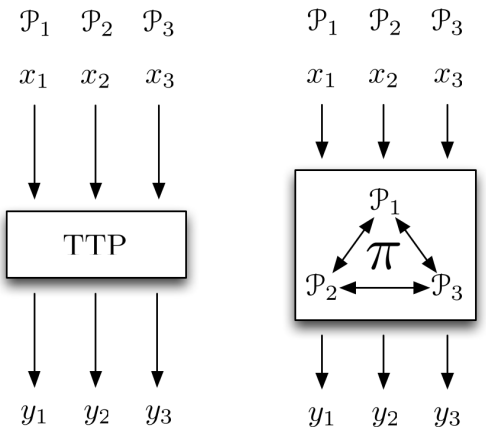
2.4 No Single Point of Trust

This section explains the motivation behind eliminating the single point of trust in the System. We identify potential sources of single point of trust in traditional statistics production. Finally, we present examples of attacks that exploit the existence of a single point of trust.

2.4.1 Trusted Third Party as Single Point of Trust

A fundamental issue in traditional statistics production is the reliance on a trusted third party (TTP) to carry out computations honestly. Theoretically, an incorruptible TTP that follows the agreed-upon logic can ensure secure computation, fulfilling all required security properties [16]. However, in practice, such a universally trusted party rarely exists. Different stakeholders may have conflicting interests, and no single entity can be assumed to be trustworthy by all. This lack of universal trust motivates the design of secure multi-party computation, which aims to eliminate the need for a TTP.

At the lowest abstraction level, MPC addresses this challenge through the ideal-real world paradigm. Instead of relying on a real-world trusted third party, MPC simulates its role using a protocol π executed by multiple parties [16, 24]. To provide security guarantees, the number of Computing Parties participating in π must be at least two. In practice, for better performance, at least three Computing Parties are often involved [3]. Figures 1a and 1b illustrate the contrast between ideal- and real-world computation models. It is important to emphasise that any actual TTP represents a single point of trust, with the potential to compromise the confidentiality and correctness of the computation.



(a) Ideal world. (b) Real world.

Figure 1. The ideal-real world paradigm [24:22]. In this case, each party simultaneously acts as an Input, Computing, and Output Party. π – protocol, $\mathcal{P}_i$ – party, x_i – input, y_i – output.

This thesis argues that the principle of avoiding a single point of trust, as established in MPC theory, must also extend to higher abstraction layers. The System, implemented as an MPCaaS solution, builds upon an existing MPC framework abstraction and must not weaken its security guarantees. Each additional abstraction layer – moving from protocol to framework to service – increases complexity and introduces new design challenges. Therefore, careful system design is required to ensure that theoretical guarantees of MPC are preserved in practice. Violating the security assumptions at any level undermines the integrity of the entire computation. To be secure, the System must avoid introducing any single point of trust. The following section presents examples of potential SPoT-threats that arise when transitioning from the MPC framework abstraction to the MPCaaS level.

2.4.2 Practical Meaning of Single Point of Trust

We present a selection of example threats related to a single point of trust. The following list is not exhaustive but provides an intuition of the issues the System is designed to mitigate by avoiding any single point of trust.

- **Unauthorised data reuse in Computation Task** refers to scenarios where previously submitted private data is reused in new tasks without the explicit consent of the data providers. Such situations may arise when highly privileged users can access private data or initiate tasks without appropriate checks and approvals.
- **Private data interception** becomes a risk when a single entity, such as the service provider, controls the communication channel.
- **Impersonation** of parties by the service provider can be used to access private data. Such an attack becomes possible when a single entity operates the infrastructure of a system, and there is no mechanism to verify the identity of the parties.

The System aims to eliminate the need for ad hoc deployment while upholding strong security guarantees. To achieve this, each process within the System that may eventually lead to leakage of sensitive information must avoid introducing any single point of trust.

Nonetheless, other sources of single point of trust remain. For example, a User's browser used to access the System may constitute a single point of trust. Browsers or client machines are often targeted for compromise. Another example involves potential backdoors in hardware or operating systems introduced by vendors. These scenarios can lead to the leakage of sensitive data or other unauthorised behaviours. However, not all such sources can be mitigated purely

on the architecture level of the System. The accessible risks depend on the specific security objective of the System. We define the relevant scope more specifically in Section 2.8.1.

2.5 Architecture Design Approach

There are various approaches to architecture design of a software system. One approach is Kruchten's "4+1" view model [29], which provides a framework for complex architecture modelling through five views, whereas each view addresses the concerns of different stakeholders. These views – logical, process, physical, development, and scenarios – are designed to represent separate aspects of a system, much like how different plans (e.g., plumbing, electrical, or elevational) guide the construction of a house.

A similar approach was later introduced by Rozanski and Woods [30], which extends the concept of *views* by incorporating *viewpoints* and *perspectives*. Like Kruchten's model, their framework aims to produce clear, manageable, and comprehensible architectural descriptions of complex systems. Standard tools used to describe software architectures include Unified Modelling Language (UML) and Business Process Model and Notation (BPMN). While views represent specific solutions tailored to meet system requirements, viewpoints offer general guidance on how such views should be constructed. Rozanski and Woods define *view* as "a representation of one or more structural aspects of an architecture that illustrates how the architecture addresses one or more concerns held by one or more of its stakeholders" [30:56]. Meanwhile, *viewpoint* is "a collection of patterns, templates, and conventions for constructing one type of view" [30:58]. There are seven core viewpoints:

- context viewpoint – describes the interactions between a system and its environment, such as scope and responsibilities;
- functional viewpoint – focuses on the functionality, including functional capabilities and external interfaces;
- information viewpoint – explains how a system manages information (e.g., information purpose, usage, flow);
- concurrency viewpoint – describes the concurrent execution of functional elements (e.g., coordination between concurrent tasks);
- development viewpoint – defines the architecture supporting the development process, such as module organisation;

- deployment viewpoint – describes the deployment environment (e.g., hardware requirements) of a system;
- operational viewpoint – outlines how a system will be managed during production, including installation and monitoring [30].

According to Rozanski and Woods, system requirements addressed in one view are typically not relevant to other views [30]. However, specific requirements, mainly non-functional ones, extend across multiple views. Architectural perspectives are used to address such requirements. Perspectives encompass a set of architectural decisions to ensure a system has specific quality properties. The authors identify 10 perspectives: accessibility, availability and resilience, development resource, evolution, internationalisation, location, performance and scalability, regulation, security, and usability.

The System adopts the *Viewpoints and Perspectives* (VP and P) approach for architecture modelling. This approach is favoured for two reasons. First, it is the preferred method within the company that designs the System, as it has proven effective in similar projects. Second, it explicitly includes security as a perspective, which is our study's focus. This perspective focuses on ensuring a system can effectively control, monitor, and audit access to resources and allowed actions while also being capable of identifying and addressing security incidents [30]. In order to narrow the scope of the thesis, the emphasis is placed on the security requirement of no single point of trust within the System. Due to the complexity of this requirement, it is decomposed into more manageable components, which can be analysed using a threat modelling technique known as *attack tree* as described later.

2.6 Architectural Decision Record

Documenting architectural decisions is as important as proposing them. One effective method is using an architectural decision record (ADR) [8]. ADR captures the architectural decision and provides details about its motivation and consequences. Additionally, ADRs allow the creation of a decision log, which helps track the evolution of decisions over time, including those relevant at a particular point and those currently in effect. Various templates exist for documenting ADRs, such as Nygard's ADR, Y-Statement, Markdown ADR, and others.

2.7 Attack Tree

The requirement of no single point of trust in the System is a security requirement. Attack tree provides a helpful way to illustrate how a single entity could compromise the confidentiality

or security of the Users or their data. An attack tree is a hierarchical diagram consisting of a root node, branches, and sub-nodes [31]. The root node represents the ultimate goal of an attack, while its children correspond to intermediate sub-goals. These sub-goals can have their own child nodes, forming a tree structure. To achieve the root goal, an attacker must fulfil the leaf-level sub-goals and recursively ascend the tree. Internal nodes in the tree are classified as either OR-nodes or AND-nodes: an OR-node requires that at least one of its child nodes is satisfied, while an AND-node requires all of its children to be achieved [31].

Other threat modelling techniques include STRIDE [32], MITRE ATT&CK [33], and PASTA [34]. However, the attack tree approach is particularly well-suited for threat modelling related to the no single point of trust requirement. An attack tree focuses on a well-defined attack objective and strikes a practical balance between expressiveness and simplicity. Thus, an attack tree is reasonable for analysing a specific system requirement rather than evaluating all potential threats.

2.8 Overview of the System

In this section, we provide an overview of the analysis [27] and architecture [10] of the System. Both documents are essential for understanding the System, including its assumptions, requirements, stakeholders, processes, and architecture. Although the complete content of these documents exceeds the scope of this study, we emphasise the elements most relevant for understanding how the System addresses the requirement of no single point of trust. Specifically, we focus on four architectural views that are most relevant to this security objective: the context, deployment, functional, and information views. While the concurrency, development, and operational views are important from the architecture analysis standpoint, they are not as significant from the security perspective.

2.8.1 System Analysis

The System is conceived as an MPCaaS solution that enables secure computations on-demand, involving multiple organisations as Input Parties or Output Parties [27]. The current focus of the System is on operation within the European Statistical System. Specifically, the System aims to support National Statistical Institutes and their partners (collectively referred to as Clients) in collaboratively defining and executing Computation Tasks. The System is designed to provide functionality for specifying the details of each Computation Task and executing it through a user-friendly interface. This includes determining the identities of the Input Parties, the format of the data to be submitted, the computation function to be executed, the nature of Output Data

and the Output Parties' identities. Each Computation Task involves several processes such as Input Data preparation, data upload, validation of uploaded data, secure computation execution, and result retrieval by Output Parties. A Computation Task involves at least two Input Parties, a minimum of three Computing Parties, and one Output Party.

The System architecture follows a principle of plane separation [27]. This principle contributes to the System maintainability and categorisation of its functions and responsibilities. The architecture is structured into three distinct operational planes.

1. The Management Plane is responsible for the System management functions such as Members onboarding, System deployment and maintenance, auditing, and the enforcement of global policies.
2. The Control Plane manages the lifecycle of Computation Tasks, including the Computation Task negotiation, Input Data protection (e.g., secret sharing), protection removal from Protected Output Data, and Restricted Data erasure.
3. The Data Plane provides functionalities required for Computation Tasks execution using MPC.

Each Plane encompasses a defined set of responsibilities, and Users are expected to interact with the Planes according to the processes relevant to their roles. The mapping of System roles to the Planes is depicted in Figure 2. The Management Plane is utilised by the System Operator for managing the System and by the System Auditor for conducting audits. The Control Plane enables Members and the System Operator to collectively negotiate and approve a Computation Task Specification and produce a corresponding Computation Task Agreement. The Data Plane is responsible for execution of a Computation Task following the details agreed upon in a Computation Task Agreement.

There are several external entities identified, providing services necessary for the operation or usability of the System [27]. For instance, Computing Parties may rely on public cloud infrastructure offered by cloud service providers. The System uses trusted execution environment (TEE) technology to enhance security. Relevant hardware and software are supplied by TEE vendors. The software vendor responsible for delivering the System is also an external entity. Another example includes contractors employed by Input Parties or Output Parties to support specific activities, such as reviewing a Computation Task Specification or parts of it.

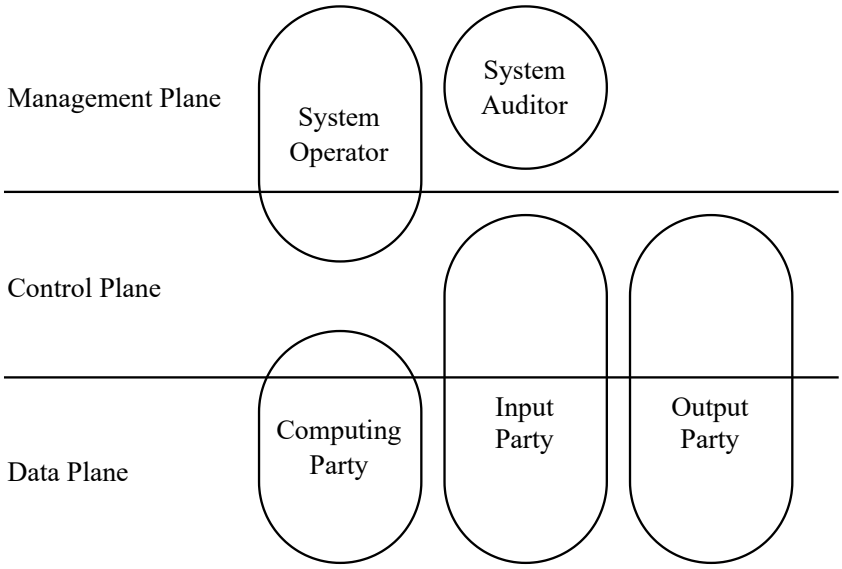


Figure 2. Mapping of System roles to the Planes of the System [27:13].

The System leverages multiple mechanisms to enhance its security [27]. These include onboarding procedures for Members, such as identity verification, credential validation, and compliance check. The System also allows the System Operator or the Input Parties to interrupt an ongoing Computation Task and delete associated Restricted Data. To reduce residual risks concerning dishonesty of Computing Parties, each Input Party may also act as a Computing Party, provided the respective onboarding requirements are met. Additional safeguards protect against various attacks, including impersonation of parties and compromise of the Computing Node software. All Members must also adhere to the System Agreement, which defines acceptable use of the System. Moreover, the System enforces policies of two types: system-wide and computation-task-specific. For instance, data lifecycle policies regulate how long Restricted Data may persist and define when it should be deleted or rendered permanently illegible. Furthermore, the MPC and TEE technologies used in the System are intended to be modular, subject to be chosen based on required performance, security, or functional properties. This design ensures System evolvability, prevents vendor lock-in and provides flexibility for Clients in selecting appropriate technologies on a case-by-case basis. The System is meant to utilise hardware-based TEEs to offer complementary security guarantees to the software-based MPC technology.

Despite the application of multiple security measures, the primary security goal is that "breaking the System must not be any easier than breaking the data sources, i.e., the Input Parties" [27:23]. This security goal defines the scope of our research, meaning that the System must not introduce **any new** single point of trust. The analysis also outlines possible points for an attack.

1. **Data input and output** – the System must not allow any party, including Computing Parties, to access Input Data or Output Data in plain text.
2. **Data storage** – the System must not store Restricted Data in a form that allows a single entity to decrypt it and compromise its confidentiality.
3. **Computation Task Specification** – an incorrect computation function or a malicious MPC technology may lead to information leakage, which must be prevented.
4. **Computation Task execution** – data must remain protected throughout the computation and not be processed in an unencrypted form. The System must allow Clients to verify that the correct Computation Task is executed using the correct MPC technology.
5. **Identity management** – the System must enable Clients to detect impersonation of a Member.

The analysis also refers to potential attacks targeting the availability and integrity of the System, but these fall outside the scope of this study. The risk of cryptographic vulnerabilities is also considered. This risk is mitigated through the modular architecture of the System, which allows the replacement of insecure components.

We also present an overview of the Computation Task lifecycle. The lifecycle consists of five main phases as follows [27].

Computation Task initiation

The first phase is Computation Task initiation. In this phase, Members collaboratively create a Computation Task Specification containing human-readable and machine-readable fields. The Computation Task Specification includes the following elements.

1. Selected MPC technology, which specifies the MPC engine to be used for the computation.
2. Client-to-role assignments, which define the responsibilities of each Client within the Computation Task.
3. Assignment of Computing Parties, which identifies the entities selected to perform the computation.
4. Cryptographic identities of the Peers (i.e., of Members participating in a Computation Task), specifying who is involved in the computation.
5. Legal contract, which outlines any additional legal constraints relevant to the data analysis.

6. Computation Task lifecycle deadlines, including 1) the deadline for signing the Computation Task Specification, 2) the deadline for Input Data preparation and upload, 3) the computation timeslot, and 4) the data retention period.
7. Data quality assurance algorithms, which define MPC programs used to verify that the Input Data meets the specified quality requirements. The computation cannot proceed unless these quality checks are successfully passed.
8. Algorithm, which determines how Input Data will be transformed into Output Data.
9. Input Data table models, which describe the structure of the Input Data, including the arity and data types.
10. Input Data table to Input Party assignments, which specify which Input Parties are responsible for providing each Input Data table.

The Computation Task is initiated once all these fields are filled in. The Computation Task initiation is followed by its consolidation.

Computation Task Specification consolidation

The drafted Computation Task Specification is a subject for review by all Peers. Once the review is complete and no further modifications are necessary, the Peers formally approve the Computation Task Specification by signing it. This process forms a legally binding Computation Task Agreement, applicable to all Peers. Once all required signatures, including that of the System Operator, have been collected, the finalised Computation Task Agreement is delivered to the specified Computing Parties.

Input Data upload

During the next phase, Input Parties prepare their Input Data following the requirements defined in the Computation Task Agreement. This preparation may include local validation procedures to ensure compliance with the specified data format. Once the Input Data is validated and consistent, the Input Parties apply appropriate protection mechanisms, such as secret sharing. The resulting Protected Input Data is then uploaded to the Computing Parties specified in the Computation Task Agreement. Upon successful upload, the Computing Parties execute the agreed-upon data quality assurance algorithms using the specified MPC technology.

Computation Task execution

Once all Input Parties have uploaded valid Input Data, the Computation Task proceeds to the execution phase. In this phase, Computing Parties execute the Algorithm specified in

the Computation Task Agreement. Execution may take up to several days, depending on the complexity of the Algorithm and the volume of Input Data. The status of the ongoing Computation Task can be monitored through the telemetry features to be provided by the System. Additionally, Clients may interrupt the execution if they detect a potential breach of their Input Data confidentiality.

Output Data retrieval

Finally, once the computation completes successfully, the authorised Output Parties retrieve the resulting Output Data. Once all Output Parties have retrieved their Output Data, or once the corresponding deadline passes, all data related to the Computation Task is deleted.

Throughout the Computation Task lifecycle, the System logs key events associated with state changes of the Computation Task [27]. In addition, the System maintains system-wide logs that are not tied to any specific Computation Task. These logs are preserved to ensure transparency and auditability, and are accessible to the Peers and/or the System Auditor.

2.8.2 System Specification and Architecture

This section focuses on technical details of the System. Concerning the deployment view, the System is decentralised and jointly operated by the System Operator and the organisations participating in computations [10]. Decentralisation is achieved by federating the System: both the System Operator and each organisation deploy and manage their own Service Nodes, collectively forming the System. A minimal deployment of the System requires a Service Node operated by the System Operator, at least one Service Node per Member, and a minimum of three Computing Nodes (each with a headless Service Node instance). Each Service Node includes a web-based interface for Users (except for headless deployments) and essential components of the Planes. This setup enables Users to perform tasks such as initiating a Computation Task, signing a Computation Task Agreement, uploading Protected Input Data, and retrieving Protected Output Data. For Computing Parties, the Service Node is required for a Computation Task deployment, which includes the initialisation of a virtual machine. This virtual machine is initialised from a public image file that contains components such as the MPC engine, the Computation Task Orchestrator (responsible for enforcing policies and Computation Task coordination), and TEE-specific modules. The specific image to be used for a Computation Task is defined in the corresponding Computation Task Agreement.

Recalling the principle of plane separation, the System consists of the Control, Data, and Management Planes. Accordingly, concerning the functional view, the System comprises

three corresponding subsystems [10]. Users interact with these subsystems via a shared Portal component, which provides a unified interface to the functionalities. Each subsystem includes multiple components, which are outlined in the following paragraphs.

Control Plane subsystem

The Control Plane subsystem provides the functionalities required to organise Computation Tasks. These include creating, processing, and distributing various data elements that form part of a Computation Task Agreement or relate to the Computation Task lifecycle. These data elements are referred to as *Computation Task artefacts*. Each Service Node hosts the Artefact Manager component responsible for creating, processing, and validating artefacts against the System policies. Once artefacts are prepared and signed, the Deployment Operator component handles the deployment of the Computation Task, and initiates the provisioning of virtual machines. This component also enables task interruption when requested by an Input Party or the System Operator. The lifecycle of a deployed task is managed by the Computation Task Orchestrator, which is local to each Computing Party. It operates based on system-wide and task-specific policies, the task state machine, and coordination with other involved Computing Parties. The Control Plane also includes the IO Service component, which enables protection of Input Data, uploading of Protected Input Data, retrieval of Protected Output Data, and transforming it into Output Data. This component belongs to the Control Plane because it enforces access control, ensuring, for example, that an Input Party cannot retrieve Output Data. The Computation Task Orchestrator interfaces with the Protected Data Store component, which handles the storage, access, and deletion of Protected Data linked to a specific Computation Task. Finally, the Telemetry Aggregator component enables Users to monitor the status of tasks in which they are involved. It also issues notifications when manual intervention is required.

Data Plane subsystem

The Data Plane subsystem comprises the following four components [10].

1. MPC engine – executes MPC programs written in a human-readable domain-specific language.
2. TEE platform – guarantees the data and code integrity and provides mechanisms for remote attestation of the deployed Computation Task and its associated software components.
3. Infrastructure platform – an environment-specific component responsible for provisioning and configuring virtual machines and associated resources, typically offered by a cloud service provider.

4. Image repository – a repository containing identifiable and System-wide known virtual machine images, each of which encapsulates the components required to execute a Computation Task.

The Data Plane subsystem is responsible for provisioning virtual machines for executing Computation Tasks and for enabling remote attestation of both the Computation Task Agreement and the software components deployed on Computing Nodes. This subsystem also executes a Computation Task [10].

Management Plane subsystem

The Management Plane subsystem governs and configures the operational aspects of the System. System-wide policies are defined and enforced via the System Policy Engine component, which also functions as the policy decision point. The Role and Access Manager component manages the association of Users, system roles, and Service Nodes. In the System, Service Nodes are identified by digital certificates, which are managed by the Node Identity Manager component. This component stores certificates and supports operations dependent on them, such as the authentication of Service Nodes. In order to enable inter-node communication, the Connection Manager component maintains mappings between IP addresses and Service Nodes, thereby establishing the network topology. This identity-based communication infrastructure is utilised by the Service Broker, which handles the routing of messages between adjacent nodes or local components. Finally, the Key Management Service is responsible for securely storing private keys and provides digital signing capabilities required by other local components within the System.

Having familiarised ourselves with the functional view, we proceed to the information view. Concerning the latter, several data elements and information flows must be considered and are described below [10].

Computation Task Agreement

A Computation Task is defined by a Computation Task Agreement, whose lifecycle is depicted in Figure 3. The lifecycle consists of five states, as described below.

1. **Emerging** – in this phase, a Computation Task is being negotiated and formulated in the form of a Computation Task Specification, which is subsequently distributed to the Input Parties, Output Parties, and the System Operator. Peers then review and sign the Computation Task Specification. Once all required signatures are collected, the System Operator must approve or reject the Computation Task Agreement.

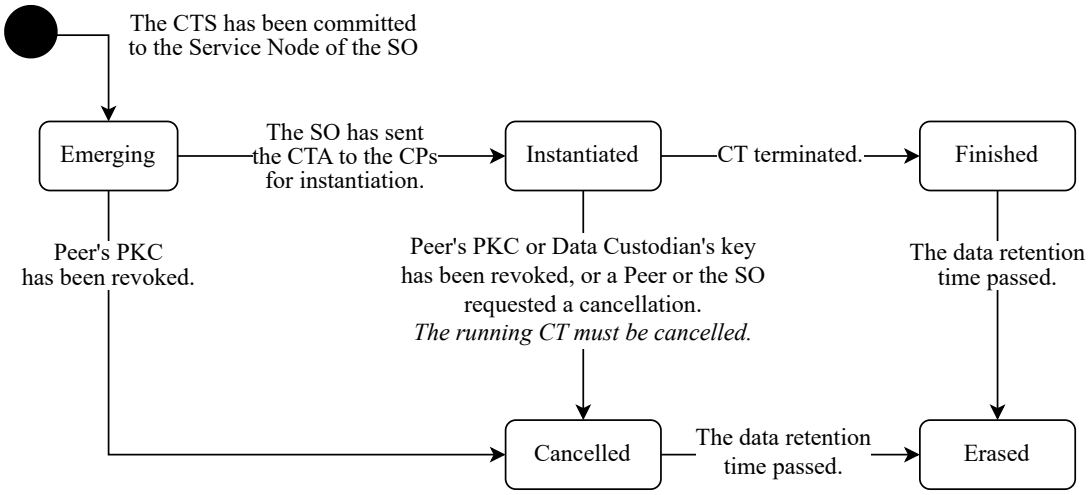


Figure 3. Lifecycle of a Computation Task Agreement [10:50]. Abbreviation designations: CT – Computation Task, CTS – Computation Task Specification, CTA – Computation Task Agreement, SO – System Operator, CPs – Computing Parties, PKC – public key certificate.

2. **Instantiated** – upon finalisation, the Computation Task Agreement is delivered to the designated Computing Parties. Based on the Computation Task Agreement, these parties prepare the necessary infrastructure for executing the Computation Task. Once the infrastructure is in place, the Input Parties upload their Input Data, the computation is executed by the Computing Parties, and the Output Parties retrieve the resulting Output Data.
3. **Cancelled** – this state occurs if the Computation Task is cancelled during or prior to execution. No Output Data is produced or returned in this case.
4. **Finished** – this state indicates the successful completion of a Computation Task, with Output Data made available to authorised Output Parties.
5. **Erased** – this is the terminal state of all Computation Tasks. Once the data retention period has expired, all data related to the task is irreversibly deleted.

The structure of the Computation Task Specification and Computation Task Agreement is complex and has been described in Section 2.8.1. Recalling the corresponding part, a Computation Task Specification consists of a core section and multiple addenda, each comprising various fields, such as the selected MPC engine, the source code of the Algorithm, the identities of the Peers, and others.

Service Node certificates

Each Service Node possesses a unique public key certificate to enable various certificate-based operations, including authentication, overview of Peer identities, asynchronous communication, and access control.

Data Custodian's and Data Analyst's asymmetric authentication key pairs

Input Parties (Data Custodians) and Output Parties (Data Analysts) use asymmetric key pairs to authenticate themselves when interacting with a Computation Task, as well as when querying Computation Task artefacts hosted on other Service Nodes.

Signatory's certificates and signatures

To formally approve a Computation Task, each Peer must digitally sign the associated Computation Task Agreement. For each signature to be legally enforceable, the System relies on certificates of identification documents and the Signatory's (digital) signature. During the review of a Computation Task Agreement, the System Operator is responsible for verifying the validity of all provided certificates and signatures based on data collected during the onboarding process.

Local user management system

The System supports integration with local user management systems (e.g., LDAP). Such functionality allows Clients to map local users and their permissions to the roles defined within the System.

Virtual machine addresses

Upon instantiation of a Computation Task, virtual machines expose network endpoints to enable interactions, such as Input Data uploads or remote attestation of the correct virtual machine image and task execution. The System disseminates the IP addresses of all such virtual machines to the relevant Peers involved in the Computation Task.

Logs and telemetry

The System collects several types of logs, namely:

1. Service Node logs,
2. logs from virtual machines executing Computation Tasks, and
3. telemetry data on state transitions of virtual machines.

The first two types are intended to be reviewed by the System Auditor to detect potential security breaches or policy violations. The third type provides visibility for Data Custodians and Data Analysts into the execution of their respective Computation Tasks. These logs may contain various information, including Computation Task artefacts, IP addresses, Computation Task Agreements, and task state transitions.

2.9 Related Work

Several MPC frameworks have already been discussed. Some of these are highly sophisticated, offering extended functionalities that, to some extent, resemble aspects of MPCaaS, and are called MPC platforms [14]. Such functionalities may include workflow automation or process support; sometimes, these framework extensions do not require ad-hoc deployments. In the following, we provide an overview of these platforms and explain why, despite their advanced features, they do not align with our understanding of MPCaaS.

For instance, Sharemind MPC allows organisations to deploy and manage the platform on their own infrastructure, while also offering a cloud platform that supports on-demand computations [35]. Other examples include Roseman Labs [19], XOR [20], and CipherCompute [36]. These systems also offer services such as specifying, approving and running computations, uploading Input Data, and downloading Output Data.

However, there are two main limitations associated with these platforms. First, some introduce a single point of trust, for example, by placing all Computing Nodes under the sole control of the service provider. Second, they are typically tied to a specific MPC technology stack, resulting in vendor lock-in, which also constitutes a single point of trust concerning the software vendor.

Another notable case is Partisia Blockchain [28], which enables service consumers to lease Computing Nodes for executing Computation Tasks. Partisia functions more as a computational marketplace than a general-purpose MPCaaS platform.

In summary, while these platforms partially overlap with the concept of MPCaaS, they lack the flexibility of technologies to be used in computations or fail to circumvent a single point of trust. The JOCONDE System is designed to fill this gap.

2.10 Summary

This chapter answered the research question RQ1: “How can architectural decisions be formulated to design the System with no single point of trust?”. We began by introducing

the layered abstraction of MPC, covering its theoretical foundations, protocols, frameworks, and the idea of MPCaaS. Then, the concept of a single point of trust was analysed, exploring its theory and practical implications, emphasising trusted third party as privacy vulnerability. We also presented the concepts *viewpoints* and *perspectives* employed in the JOCONDE project for the architecture design. The use of architectural decision records supported this approach. All this information enables a structured threat modelling process using an attack tree approach to assess and mitigate risks. We also gave an overview of the resulting analysis and architecture of the System. Finally, we reviewed related work to position the System within the field of existing MPC technologies.

3. Method

This chapter outlines the research method employed to achieve the objectives of our study. The utilised method is depicted in Figure 4[5].

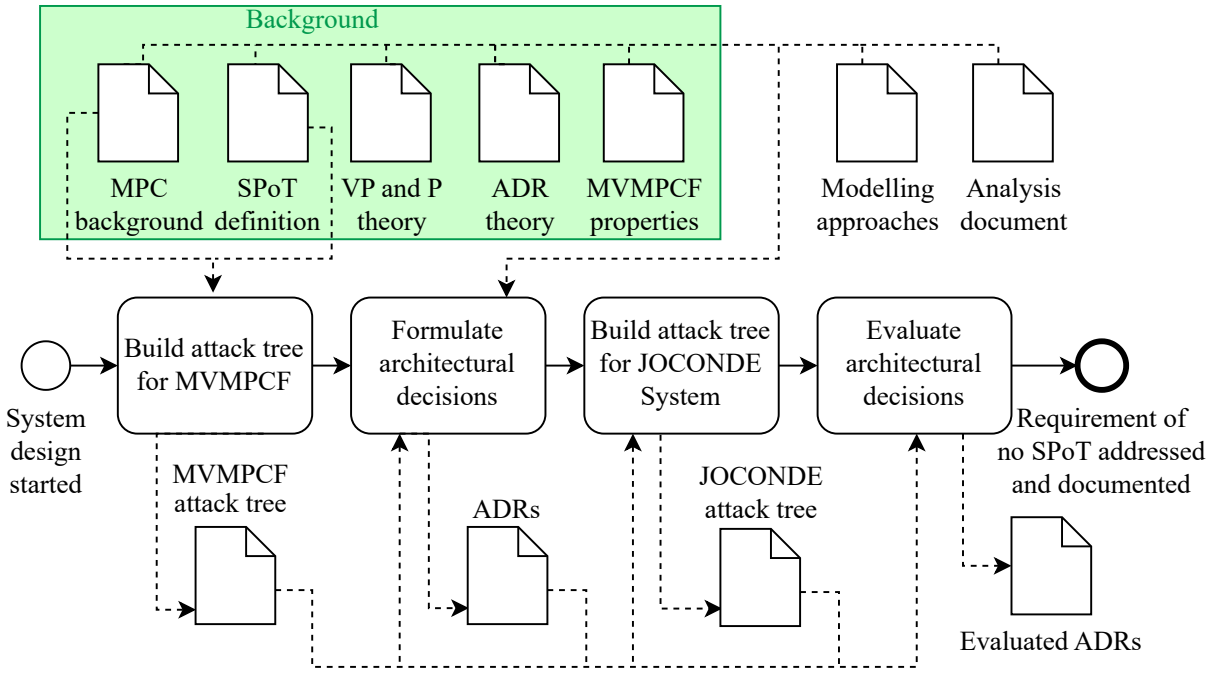


Figure 4. Used research method.

At a high level, the method consists of four main steps. First, we build an attack tree for the Minimum Viable MPC Framework (MVMPCF). Second, we formulate architectural decisions that eliminate sources of single point of trust. Third, we revisit the MVMPCF attack tree and complement it with the security controls and specifics of the System. Fourth, we evaluate the architectural decisions. The following sections elaborate on each step of the method in more detail.

3.1 Building Attack Tree for Minimum Viable MPC Framework

The requirement of no single point of trust in the System is too abstract and cannot be addressed directly. Although more specific system requirements were introduced in the corresponding

[5]In this figure, the file-like icons represent artefacts. These artefacts are either used or produced throughout the phases of the study. With respect to activities, artefact usage is illustrated by an incoming arrow, while an outgoing arrow indicates artefact production.

deliverable [27], they lack sufficient detail to guide architectural decisions to avoid a single point of trust. As a result, more precise requirements are necessary.

This thesis focuses on a specific security requirement. If an attacker were to target a single point of trust in the System, their objective would be to *compromise the confidentiality of a Client's sensitive data*. This goal is used as the root of the attack tree. From this root, we recursively identify the sub-goals an attacker would need to achieve, refining our understanding of potential threats.

We begin with constructing an attack tree for a Minimum Viable MPC Framework. This approach enables us to identify the potential threats associated with a single point of trust, even at a high level of abstraction. As the Minimum Viable MPC Framework is abstract, analysing threats that depend on system implementation is impossible. Nevertheless, examining an abstract system highlights the threats that must be considered in depth when designing a particular implementation.

Given the vast – potentially infinite – number of attack scenarios, the attack tree may not capture every possible method of exploiting a single point of trust. Nevertheless, significant effort was invested in collaboration with project stakeholders, security engineers, and researchers to ensure comprehensiveness. Feedback was gathered iteratively throughout the process through regular meetings to address any gaps or omissions in the attack tree. Section 4.1 presents the built attack tree and conclusions.

3.2 Formulating Architectural Decisions

This step focuses on formulating architectural decisions based on the previously constructed attack tree. We begin with an analysis of the threats represented by the nodes of the attack tree. Then, we formulate corresponding architectural decisions to render the identified attacks infeasible. The formulation of architectural decisions consisted mainly of brainstorming, discussions with the project stakeholders and reviews by the security engineers. Ideally, such decisions target nodes closer to the tree's root, as mitigating these can prevent multiple downstream threats. Nevertheless, a single high-level decision may be insufficient for complex attack paths, requiring multiple measures. Accordingly, while the analysis begins at the root, some architectural decisions can be formulated only at deeper levels of the tree and, in some cases, at the leaf nodes. A low-level architectural decision is formulated for every node whose mitigation contributes to avoiding a single point of trust. Given that the architectural design

employs viewpoints and perspectives, each decision is reflected across the architectural views. We leverage existing solutions from prior work in MPC or general software design practices wherever possible. For proper documentation, we use ADRs, following the template proposed by Nygard [37]. According to this template, each ADR consists of the following elements:

1. title – a brief noun phrase that summarises the architectural decision;
2. context – a description of the problem or requirement that motivates the decision;
3. decision – a detailed description of the architectural change or solution;
4. status – a current state of the decision, such as *accepted*, *rejected*, or *superseded*;
5. consequences – a reflection on the positive and negative implications of the decision [37].

The collection of these ADRs constitutes a set of architectural artefacts. These artefacts are input for the evaluation step and presented in Section 4.2.

3.3 Building Attack Tree for JOCONDE System

After formulating architectural decisions, we revise the previously constructed attack tree and incorporate threats specific to the System with the corresponding security controls. The attack tree is built based on the System analysis [27] and architecture [10], summarised in Sections 2.8.1 and 2.8.2, respectively. The tree describes potential attacks involving such entities as the System Operator, cloud service provider, and Computing Party.

While capturing every possible threat exploiting a single point of trust is impossible, the tree was refined in close collaboration with project stakeholders, security engineers, and researchers to minimise the risk of overlooking critical threats. Section 4.3 presents the resulting attack tree and conclusions.

3.4 Evaluation

The final step involves validating whether the System is free from a single point of trust. Key details required for validation are documented in the produced ADRs, including the architectural decisions and associated consequences. These records clarify how each decision mitigates specific attacks that rely on the existence of a single point of trust. To facilitate the evaluation, we also provide an explicit mapping between the ADRs and the nodes of the attack tree, as discussed in Section 3.3.

We validate the results by interviewing three independent experts who are not involved in the writing of this thesis. Each interview focuses on validating a specific aspect of the results, has its own goal, and is conducted in order as follows.

1. **Completeness of threats** – to confirm that no relevant threats related to a single point of trust are overlooked.
2. **Sufficiency of mitigations** – to evaluate whether the formulated architectural decisions effectively mitigate the identified threats.
3. **Business feasibility** – to verify that the measures align with the business requirements of official statistics.

Presentation slides support interviews. The slides depend on the content of the discussion. Each interview is structured into three distinct phases. In the first phase, the interviewee is provided with an introduction to the project (if necessary), the goal of our study, and a brief overview of the method used to achieve that goal.

The second phase is a free-form discussion, the content of which varies according to the specific property being validated as follows.

In order to assess the completeness of our threat model, we iteratively present the identified threats with the respective mitigation measures to the interviewee, an expert in MPC technologies. In each iteration, the interviewee is encouraged to examine the analysis, attempting to identify overlooked attack strategies or flaws in reasoning. The interview focuses on answering the following questions.

1. What SPoT-related threats, if any, have not been identified in the current analysis?
2. Which of the identified SPoT-related threats are insufficiently mitigated?
3. Are there any mitigation measures that eliminate one SPoT-threat but inadvertently introduce a new, unaddressed one?

A similar approach is used to evaluate the sufficiency of the formulated mitigation measures. In contrast to the first interview, we iteratively present each mitigation measure first, followed by the specific SPoT-related threats it is intended to address. Each iteration is followed by a discussion, during which the interviewee – a cybersecurity expert – assesses whether the formulated measure is adequate. The following questions guide the discussion.

1. Do the formulated measures sufficiently mitigate the identified SPoT-threats?
2. Are there alternative approaches that would more effectively address these SPoT-threats?

The business feasibility is validated through an interview with an expert in official statistics. The validation focuses on the impact of the applied measures on business requirements. Effects of architectural decisions are grouped into four categories: 1) infrastructure, 2) software/artefact verification, 3) identities, and 4) user interactions. Each category is presented individually, and the interviewee is asked to assess whether any of these effects conflict with the project's business requirements. The central question for the interview is whether architectural decisions are compatible with the operational and regulatory needs of the official statistics production.

The third part of each interview serves as a concluding phase. During this phase, we revisit any unanswered questions from the earlier discussion. Additionally, the interviewee is invited to provide general feedback on the identified threats and formulated mitigation measures. This phase captures further insights or concerns that may not have surfaced during the main discussion. The outcomes of the validation process are presented in Chapter 5.

3.5 Threats to Validity

We acknowledge several threats to the validity of the results obtained through the proposed method. The first threat concerns the subjectivity involved in threat elicitation, attack tree construction, and the formulation of architectural decisions. These activities depend heavily on the author's judgment, which introduces potential bias and limits the replicability of the results. Such subjectivity may lead to blind spots, overemphasis on familiar threat patterns, or disproportionate focus on specific aspects of a system. To mitigate this, we iteratively engaged project stakeholders throughout the process to review and refine the analysis and decisions.

The second threat relates to the validation process based on interviews with three experts. Although their feedback was insightful and valuable, a larger and more diverse group of interviewees could have provided additional perspectives and enhanced the generalisability of the findings. This threat is also partly mitigated by the continuous involvement of project stakeholders, who provided feedback on both the methodology and intermediate results. Nonetheless, the potential influence of the interviewer during the expert interviews remains a source of bias. The presence and guidance of the interviewer may have unintentionally shaped the participants' responses and thus affected the objectivity of the validation outcomes.

3.6 Summary

This chapter details the method used to reach the study's goal, and is structured into four phases. The first phase involves constructing an attack tree to model how an adversary might exploit a single point of trust in the Minimum Viable MPC Framework to compromise user data. In the second phase, architectural decisions are formulated by analysing the threats identified earlier and accounting for the specifics of the System. The decisions mitigate the SPoT-threats and are documented using ADRs. Third, we revisit and supplement the previously built attack tree with these decisions. The final phase validates the threat model and the decisions. The evaluation is conducted through expert interviews, focusing on three aspects: completeness of threats, sufficiency of mitigations, and business feasibility. The following chapter presents the results of the application of this method.

4. Results

This chapter addresses the research question RQ2: "What architectural decisions enable the design of the System with no single point of trust?". We present the results of the threat modelling and the formulated architectural decisions that mitigate the identified SPoT-related threats. The chapter is structured into three main parts. First, we present the attack tree constructed for the Minimum Viable MPC Framework. This tree represents the potential threats in a baseline architecture and forms the basis for identifying vulnerabilities related to a single point of trust. Second, we formulate the architectural decisions mitigating the identified threats, considering the specifics of the System. Third, we revisit the attack tree to reflect the made architectural decisions. The final version of the attack tree illustrates how the design of the System has evolved to eliminate the previously identified SPoT-threats.

4.1 Built Attack Tree for Minimum Viable MPC Framework

The first step involved constructing a threat model for the Minimum Viable MPC Framework, using an attack tree, as described in Section 3.1. To enhance clarity and readability, the model was divided into several fragments. Nevertheless, these fragments collectively represent the complete threat model. We also provide guidance on how to interpret these diagrams. AND-nodes are explicitly marked with an icon and corresponding label. Where relevant, OR-nodes are similarly indicated with the dedicated icon. Nodes without icons are assumed to be OR-nodes.

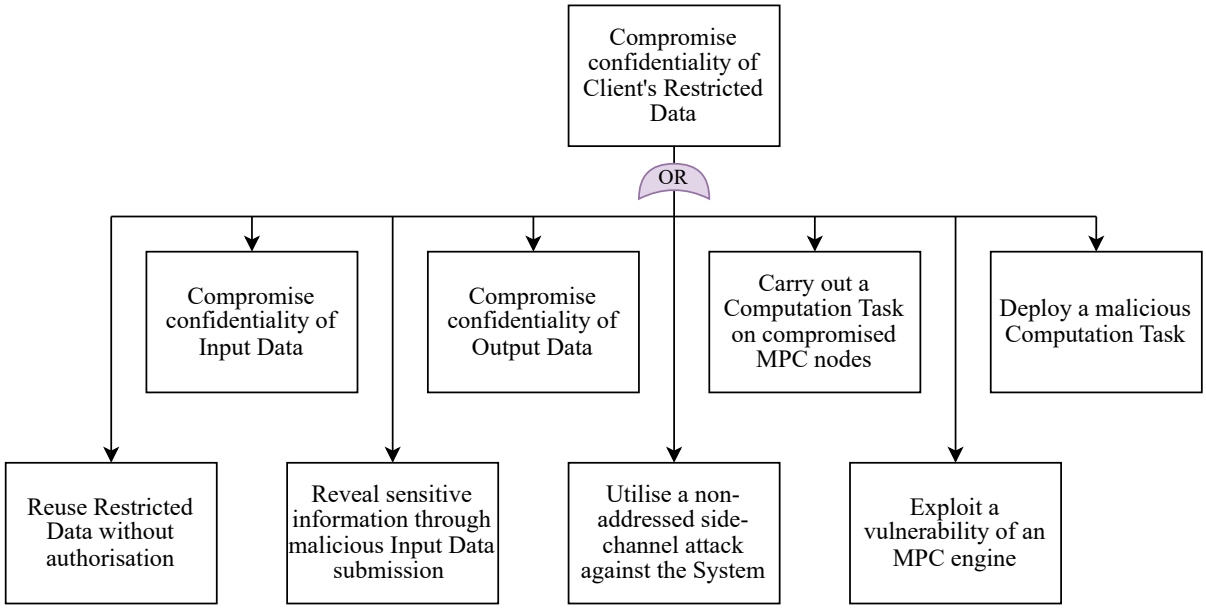


Figure 5. Overview of high-level threats that could compromise confidentiality of Client's Restricted Data.

Concerning the threat model, as shown in Figure 5, we identified eight high-level threats (sub-goals) that could potentially lead to the compromise of a Client's Restricted Data. These threats are analysed in further detail and presented as separate sub-trees. However, two threats – "Utilise a non-addressed side-channel attack against the System" and "Exploit a vulnerability in an MPC engine" – are not expanded further. This is due to the unpredictability of zero-day vulnerabilities. Nonetheless, it is important to highlight that a successful side-channel attack would require compromising all Computing Parties (or their respective communication channels) involved in a given Computation Task. Thus, there is no single point of trust involved. Regarding the vulnerabilities in an MPC engine, it is the responsibility of each Peer to ensure the trustworthiness of the engine they employ, for example, by inspecting the source code beforehand. In the following paragraphs, we analyse the remaining six threats in greater detail.

Reuse Restricted Data without authorisation

We begin with the threat of unauthorised Restricted Data reuse, which should be interpreted more broadly than merely reusing Restricted Data in another Computation Task. Here, *reuse* refers to any action involving Restricted Data that occurs outside the execution of an authorised Computation Task. As illustrated in Figure 6, two primary attack vectors are possible within the Minimum Viable MPC Framework.

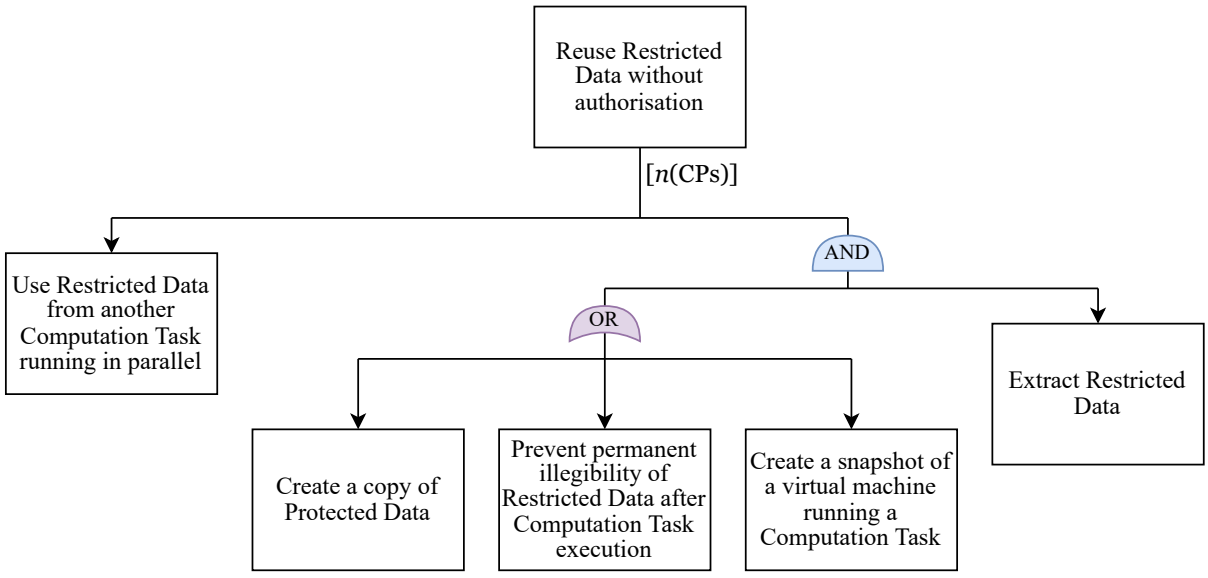


Figure 6. Attack tree for unauthorised Restricted Data reuse.

According to the definition of MPC, both scenarios require successful compromise of multiple Computing Parties[6].

1. **Use Restricted Data from another Computation Task running in parallel** – this attack is feasible if there is no protection on Restricted Data when it is in use.
2. A scenario of a **post-execution reuse** requires 1) creating a copy of Restricted Data, preventing its deletion, or generating a snapshot of a (virtual) machine containing Restricted Data, and 2) extracting legible data. The attack targets the period after a Computation Task complete. Extracting Restricted Data may range from trivial (e.g., accessing an unencrypted copy) to complex (e.g., retrieving data from a protected machine snapshot).

Concerning the SPoT-actor for such an attack, the computing infrastructure provider (e.g., a cloud service provider) could be regarded as a privileged adversary, particularly in scenarios where multiple Computing Parties employ the same service provider. It is important to emphasise that this discussion pertains to the Minimum Viable MPC Framework, which does not yet incorporate measures against party collusion (e.g., through Computing Parties selection mechanisms).

Compromise confidentiality of Input Data

Another potential attack vector involves the compromise of Input Data, as illustrated in Figure 7. We identify two scenarios. The first involves the compromise of Client software that enables Clients to interact with the system, such as to apply protection on Input Data or upload it. In this case, the MPC software vendor constitutes a single point of trust.

The second scenario considers the interception of Protected Input Data. To succeed, an adversary must compromise the communication channels between the Client and multiple involved Computing Parties. We identified two primary threats: 1) packet sniffing and 2) man-in-the-middle attacks. The privileged SPoT-actors in these cases include the Internet service provider, Internet infrastructure provider, and computing infrastructure provider (in the event of collusion among Computing Parties). This work focuses primarily on the latter, as the first two

[6]To be precise, assuming that all Computing Parties must be compromised to breach confidentiality is inaccurate. MPC engines differ in the number of corrupted parties they can tolerate, depending on their security model. Nevertheless, a successful attack requires compromising multiple parties at once, i.e., this does not introduce a single point of trust. Thus, for the sake of simplicity, the notation $[n(\text{CPs})]$ in the attack tree should be interpreted as the number of corrupted parties a given engine cannot tolerate.

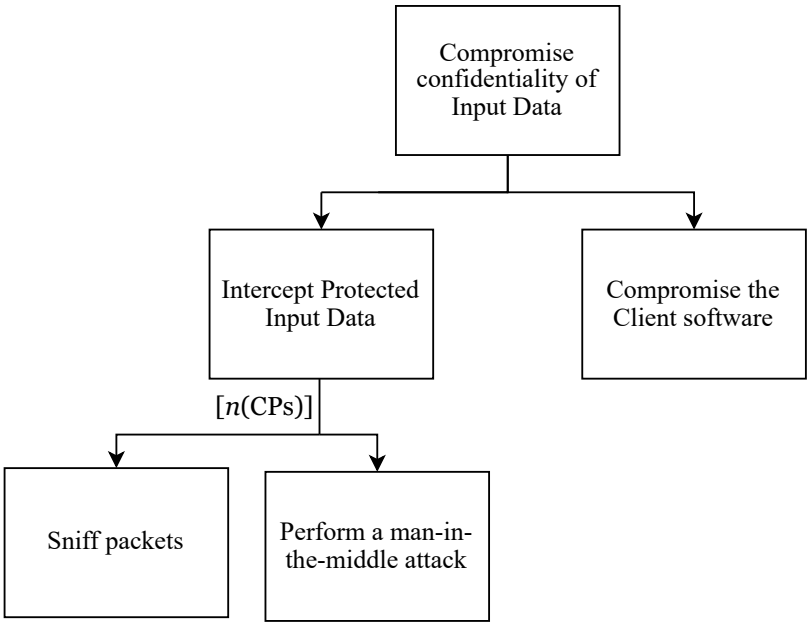


Figure 7. Attack tree for compromise of Input Data.

actors are not introduced by the System. Nonetheless, the protective measures to mitigate risks associated with the computing infrastructure provider may also reduce exposure to threats from Internet-based actors. We describe the measures in Section 4.2.

Compromise confidentiality of Output Data

Input Data is not the only confidential information considered in our threat model; the confidentiality of Output Data must also be protected. As illustrated in Figure 8, there are three scenarios in which the confidentiality of Output Data may be compromised.

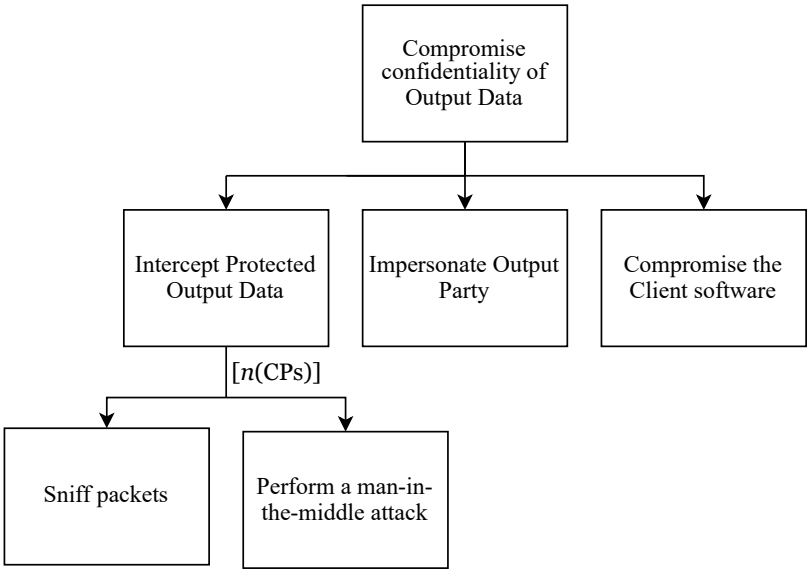


Figure 8. Attack tree for compromise of Output Data.

In the Minimum Viable MPC Framework, an attacker may impersonate an Output Party to query Output Data. The other two threats are similar to those discussed in the context of Input Data compromise: 1) interception of Protected Output Data during its transit between the Output Party and the Computing Parties, and 2) compromise of Client software used to download Protected Output Data and remove the protection. For single point of trust, advantaged attackers include providers of Internet service, Internet infrastructure, computing infrastructure, the MPC software vendor, and the service provider (e.g., the System Operator in the context of the System). The first four actors have already been discussed. However, the service provider poses a distinct threat, as it typically manages access control based on user identities and thus may act as a single point of trust in scenarios involving unauthorised Output Data access.

Reveal sensitive information through malicious Input Data submission

The confidentiality of a Client's Restricted Data may also be compromised by other peers submitting malicious Input Data. Such an attack is feasible if the MPC engine contains a vulnerability or a Computation Task is intentionally constructed maliciously. In order to succeed, the attacker must achieve two goals, as illustrated in Figure 9.

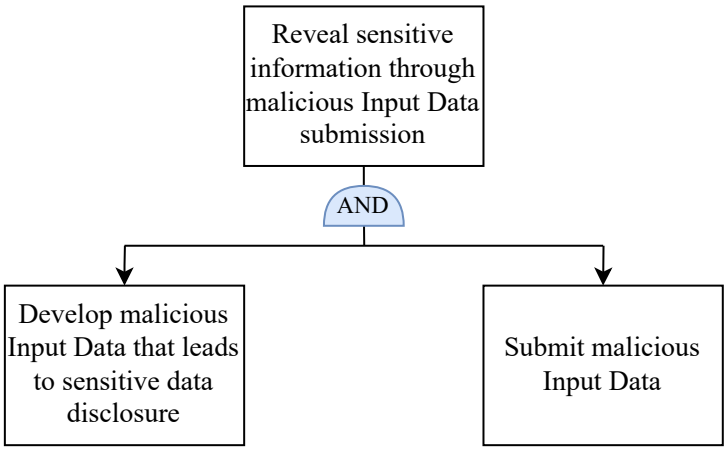


Figure 9. Attack tree for compromise of Restricted Data confidentiality through submission of malicious Input Data.

First, a malicious dataset must be crafted to cause data leakage, such as ensuring an insufficient number of records in a Computation Task that employs aggregation algorithms. Second, the malicious Input Data must be successfully submitted to the Computing Parties executing the task. This threat is particularly relevant in the context of single point of trust when only two Input Parties are involved in a Computation Task, each becoming a point of trust to each other.

Carry out a Computation Task on compromised MPC nodes

Another option to compromise the confidentiality of a Client's Restricted Data involves compromising multiple Computing Parties participating in a Computation Task, as illustrated in Figure 10.

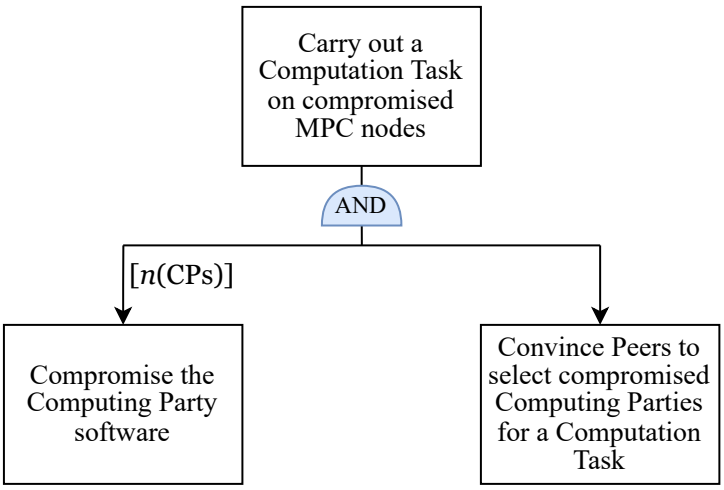


Figure 10. Attack tree for Computation Task execution on compromised MPC nodes.

The specifics of such compromise depend on the implementation of a particular system. However, a prerequisite for this attack is the successful manipulation of Peers into selecting the compromised Computing Parties for task execution. This scenario highlights multiple advantaged SPoT-actors: 1) the computing infrastructure provider, 2) an entity with control over multiple Computing Parties, and 3) MPC software vendor.

Deploy a malicious Computation Task

It is critical that Peers understand the logic of a Computation Task to ensure it does not result in the unintended disclosure of sensitive data. As illustrated in Figure 11, this attack unfolds in three steps.

1. Design a malicious Computation Task that leads to the disclosure of Restricted Data intended to remain private.
2. Submit the crafted Computation Task to all MPC nodes.
3. Convince all[7] Input Parties involved in the task to upload their Input Data.

[7]The notation $[n(\text{IPs})]$ in the attack tree should be interpreted as all Input Parties partaking in a Computation Task.

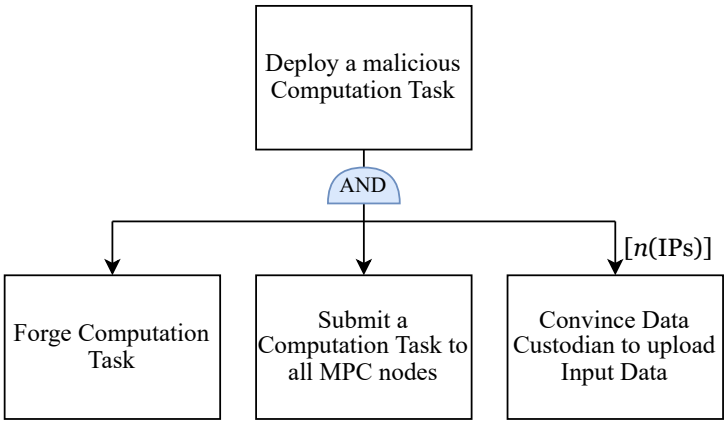


Figure 11. Attack tree for a malicious Computation Task deployment.

With respect to a single point of trust, two actors are particularly advantaged: the computing infrastructure provider and the entity submitting the Computation Task. The former becomes a single point of trust if multiple Computing Parties are hosted on the same infrastructure, while the latter – in the absence of mechanisms for verifying which Computation Task is actually to be executed on the MPC nodes.

Having identified SPoT-related threats for the Minimum Viable MPC Framework, we must consider how they affect the JOCONDE System. The System must incorporate appropriate mitigation strategies. The following section presents the architectural decisions made to mitigate these threats.

4.2 Formulated Architectural Decisions

Following the method described in Section 3.2, we formulated the architectural decisions to mitigate the threats identified in the previous step. This section outlines these decisions. Each decision is documented in an architectural decision record included in the appendices. While the ADRs follow Nygard's template, this section presents the decisions in a narrative format to enhance readability and coherence.

4.2.1 Minimum of Three Independent Computing Parties per Computation Task

According to MPC theory, a secure computation requires the involvement of at least two Computing Parties. In practice, however, it is common to involve a minimum of three Computing Parties per Computation Task [3]. To prevent breaches of Restricted Data confidentiality due to collusion, these parties must be completely independent [27]. Consequently, the first architectural decision defines that each Computation Task be executed by no fewer than three

independent Computing Parties (refer to Appendix I, ADR-1) [27]. This avoids the need to trust any single Computing Party, as the computation remains secure given that at least one participating Computing Party behaves honestly.

Architecturally, this decision primarily impacts the deployment view, as illustrated in Figure 12. Another challenge is ensuring the independence of participating Computing Parties. To address this, the System 1) enables selection of Computing Parties among the onboarded ones, and 2) allows an Input Party to also act as a Computing Party, provided they have completed the onboarding [27]. The ultimate decision to trust a specific Computing Party lies with the Input Party.

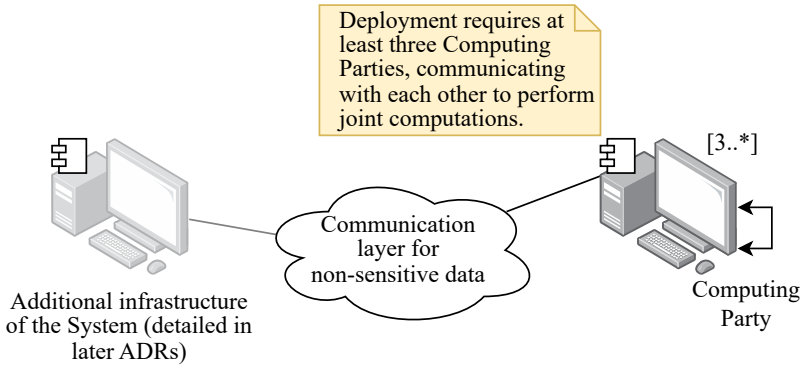


Figure 12. Deployment view of the System satisfying the requirement of three independent Computing Parties.

This decision differentiates the System from traditional client-server information systems. Whereas client-server systems typically rely on a single backend, the System requires at least three Computing Nodes to collaboratively perform a Computation Task, with communication occurring between all partaking Computing Parties. This approach aligns with established practices in MPC architectures.

4.2.2 Direct Communication Between Clients and Computing Parties

Each Computation Task execution requires transferring Input Data or Output Data between Clients and Computing Parties. Given that at least three Computing Parties are involved in a single Computation Task, one option might be to employ a reverse proxy server to simplify the data transfer process. However, such a server would introduce a single point of trust. While encrypting transmitted data could mitigate this issue, it does not eliminate the risk. Specifically, the host of the proxy server could retain the encrypted data. If vulnerabilities in the encryption algorithm are discovered, this could breach the confidentiality of Restricted Data.

To avoid introducing such risks, the System adopts direct communication between Clients and Computing Parties, as depicted in Figure 13 [10]. This approach removes the need to trust the host of a reverse proxy server (e.g., the System Operator), thereby simplifying trust building for Users of the System. The establishment of secure channels between Clients and Computing Parties is discussed further in Section 4.2.4.

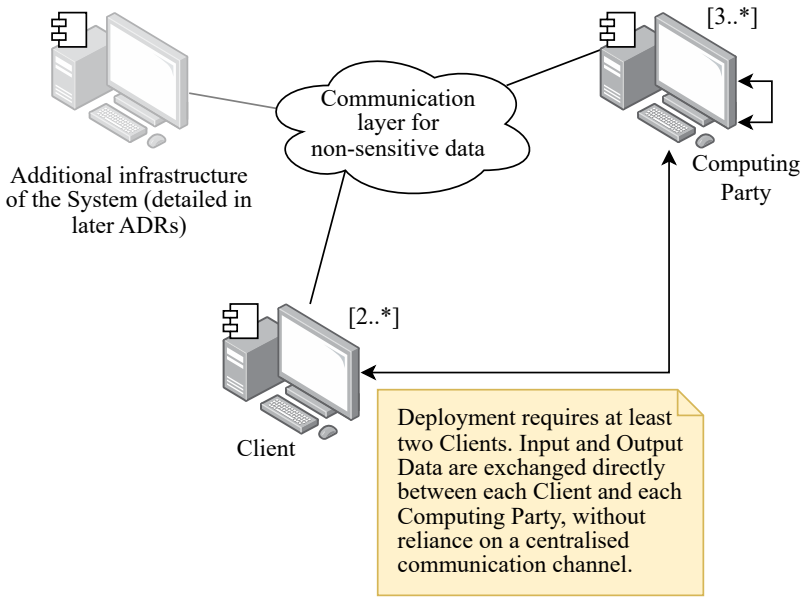


Figure 13. Deployment view of the System enforcing direct communication between Clients and Computing Parties.

Concerning the consequences of this decision, every Client must establish and maintain secure channels with multiple Computing Parties. Secure and authenticated communication channels between all parties require a key distribution and identity management mechanism. Thus, this decision reflects a conscious trade-off: increased system complexity in exchange for a more robust and decentralised trust model. The decision is documented as ADR-2 in Appendix I.

4.2.3 Services Decentralisation

The following architectural decision addresses the interactions of Users with the System. Among other functionalities, the System must enable Users to create Computation Tasks, sign Computation Task Agreements, upload Protected Input Data, and download Protected Output Data. These functionalities are provided by different components of the System. However, ensuring the System usability is necessary without introducing a single point of trust, while maintaining trust from the Clients.

The trade-off between centralised and decentralised services is closely tied to usability and trustworthiness. A centralised architecture, managed by the System Operator, would simplify the System development and maintenance. However, such an approach would make it difficult to build trust towards the System among Clients without reducing usability and maintainability.

In order to address this problem, the architectural decision is to decentralise the System services, resulting in a network of multiple Service Nodes (refer to Appendix I, ADR-3) [10]. Each Client is expected to operate at least one Service Node hosting a subset of the System services. Similarly, the System Operator operates at least one Service Node to oversee and manage the System, as illustrated in Figure 14.

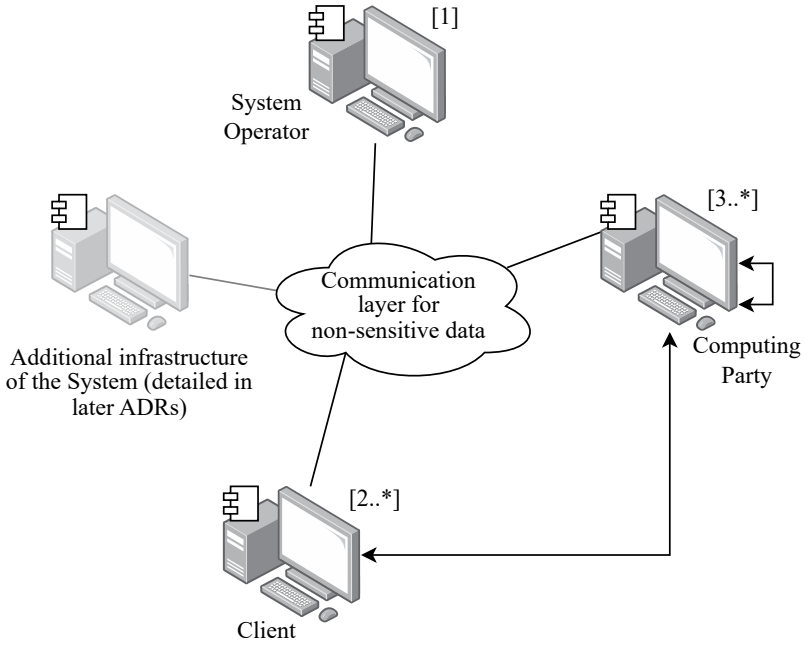


Figure 14. Deployment view of the System with decentralised services.

This decision removes the need to trust the System Operator, as Clients deploy services on infrastructure they control, thereby gaining greater oversight over interactions with external entities. However, before deployment, the components must be either open-source and reviewed by trusted entities or audited by independent auditors. Such transparency prevents the software vendor from becoming a single point of trust. From a development perspective, this approach reduces the additional mechanisms that would otherwise be required to establish trust in the System.

Nevertheless, decentralisation introduces architectural complexity, primarily due to the increased need for coordination and communication among multiple Service Nodes (as described later in

Section 4.2.5). It also increases the operational burden on the IT personnel of Clients, particularly concerning deployment and ongoing maintenance. These drawbacks can be mitigated through containerisation technologies and automated deployment tools such as scripts.

4.2.4 Combination of TEE and MPC

Several identified threats require the compromise of Computing Nodes, typically through tampering with the Computing Node software. For instance, a compromised node could execute a Computation Task without verifying signatures or could violate data lifecycle policies, potentially leading to a breach of Restricted Data confidentiality. Therefore, the System must ensure the confidentiality of Restricted Data not only during its transit but also throughout and after computation.

To address this requirement, the System employs a virtual-machine-based trusted execution environment with hardware-enforced encryption, as illustrated in Figure 15 (see also Appendix I, ADR-4) [14, 27]. This technology enables the CPU to protect data and code within the TEE from exterior attackers, including those originating from the operating system of the host machine [27].

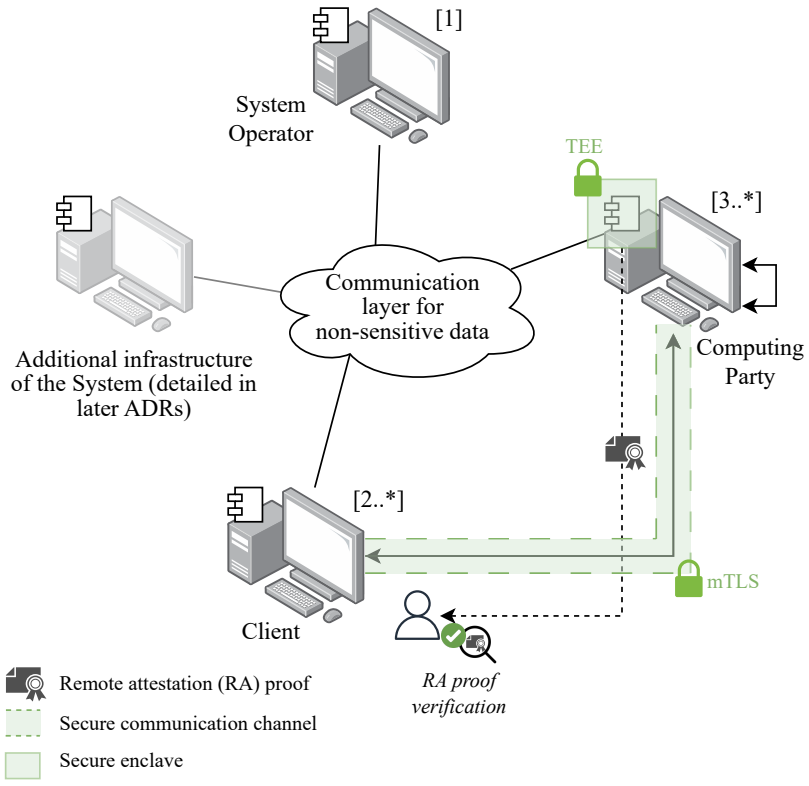


Figure 15. Deployment view of the System leveraging TEE.

Specifically, the use of TEE protects against tampering with the Computing Node software deployed on the Computing Node, encompassing both the Control Plane components and the

MPC engine. However, a TEE alone does not establish trust in the Computing Node software. To be considered trustworthy, the software must be either open-source or auditable. As discussed previously, this implies that a reputable entity or an independent auditor should review the software. Such third-party verification is essential for Clients to trust the software. Concerning the proprietary MPC technologies, an independent audit is the only viable option to be interfaced with the System.

TEEs rely on cryptographic mechanisms involving three types of keys to protect data and code [14]. The first is a symmetric key bound to the virtual machine, accessible only to the CPU and not extractable by any software or hardware. This key encrypts both memory and stored data, preventing leakage and ensuring that all information effectively becomes inaccessible when the TEE (and the key) is destroyed. The second key is used for remote attestation of software running within the TEE. A secure CPU component uses a private key to sign hashes of the software and data, producing a proof that can be verified using a corresponding public key. The proof allows external entities (e.g., an Input Party) to assess whether the execution environment is trustworthy. The third key is used for the establishment of secure communication channels between the Users and the TEE, typically using the TLS or mTLS protocol [14].

From an architectural perspective, this decision affects the functional view of the System, as shown in Figure 16[8]. Most importantly, the System must 1) utilise the Image Repository component to retrieve pre-baked Computing Node software images based on the Computation Task Agreement, and 2) provision a new virtual machine for each Computation Task using the said images. Using TEE also requires specialised hardware that supports such environments. This requirement influences the onboarding process, as only those equipped with the appropriate hardware can participate as a Computing Party.

---

[8]This component diagram depicts the interfaces the Data Plane requires and provides. The components have been described in Section 2.8.2. Concerning the interfaces, **IDataTransferProtocol** allows to protect data, remove the protection, as well as upload and download Protected Data; **IMpcActivation** – to parametrise and invoke an MPC computation; **ICapabilities** – to query the supported features of an MPC engine; **IAttestationPlatformApi** – to remotely attest virtual machines; **IVmProvision** – to provision and configure virtual machines; **ILoadImage** – to retrieve pre-built virtual machine images from the Image Repository; **IDataAccess** – to access Protected Data during an MPC execution.

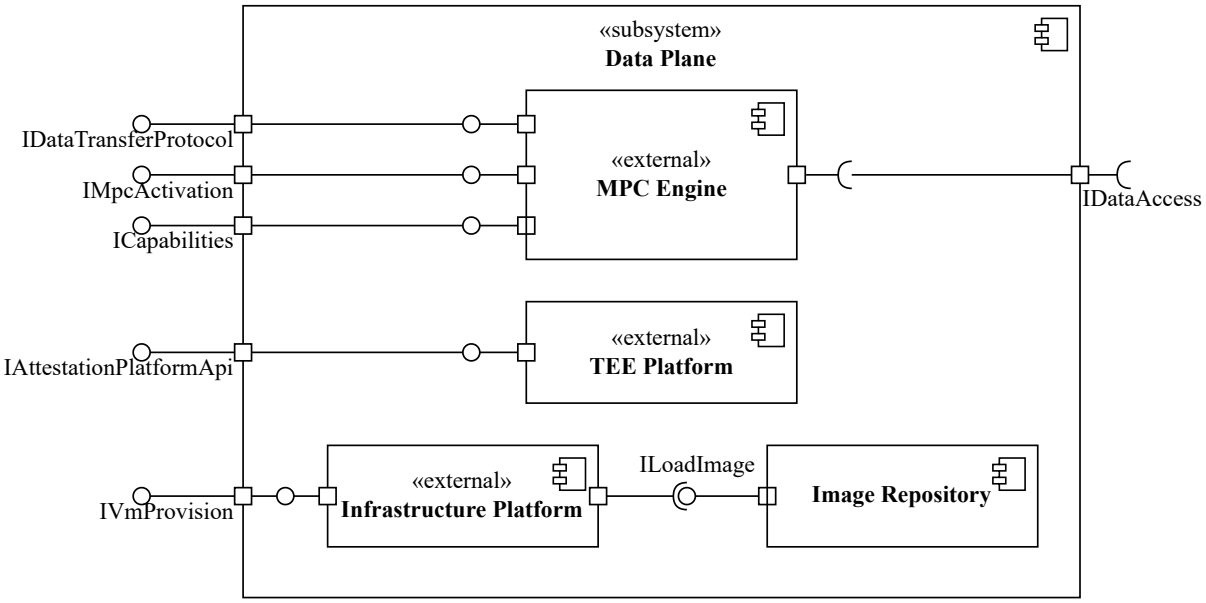


Figure 16. Functional view of the Data Plane [10:44].

However, TEEs are not without their problems. Several vulnerabilities in TEEs have been identified [38, 39]. Vulnerabilities or potential backdoors may compromise the integrity or confidentiality of data or software running within this environment [14]. In the System, this risk is mitigated by employing TEEs from multiple vendors, as the exact exploit is unlikely to be applicable for technologies from other vendors [14]. Additionally, since the System integrates both TEE and MPC, breaching the confidentiality of Restricted Data would require the compromise of multiple TEEs involved in the computation. From a trust perspective, the introduction of TEEs reduces the need for Clients to trust individual Computing Parties. Furthermore, leveraging TEEs from different vendors helps to mitigate residual risks and prevents the introduction of a single point of trust.

4.2.5 Tree Topology

The System must support usability across organisations of different sizes and internal structures. Thereby, the System accommodates multiple Client-internal stakeholders, such as Data Analyst, Data Custodian, and Signatory. For example, the Signatory may be responsible for signing the Computation Task Agreement. In contrast, the Data Custodian handles the uploading of Protected Input Data. However, in some organisations, particularly those with strict internal data protection policies, Data Custodians are not permitted to disclose Restricted Data (associated with a computation) to the organisation's broader infrastructure. To respect such constraints, the Data Custodians must be able to protect Input Data locally and upload the Protected Data directly to the Computing Nodes, avoiding exposure of data within the organisation. The same

principle applies to the download of Protected Output Data and subsequent protection removal. While the decentralisation of the System infrastructure has already been analysed, a similar mechanism is needed to support intra-organisational trust models and security requirements.

Theoretically, multiple options exist to prevent the intra-organisational transmission of Input or Output Data, as outlined below.

1. Develop a browser-based application incorporating data protection and Computation Task integrity verification components.
2. Implement a lightweight version of the Service Node software, containing only the data protection and Computation Task integrity verification components.
3. Enable each Client-internal stakeholder to operate their own full-featured self-hosted Service Node.

The first option is extremely complex from a technological aspect. The second option is technically less demanding. However, it introduces maintenance overhead by requiring the development of multiple software modules. In contrast, the third option implies no additional overhead, as the System is already designed to comprise multiple Service Nodes, as discussed in Section 4.2.3. Furthermore, integration of any information system with the System necessitates an onboarding process [27]. Regardless of the chosen approach, connecting a program or Service Node directly to the System Operator's Service Node instance would require both onboarding and certification for each new component or Service Node. This process is already established for full-fledged Service Nodes. Overall, the third option imposes the least operational and development burden, and thus it has been selected as the preferred solution.

The decision is to adopt a tree topology of Service Nodes to avoid the overhead associated with onboarding Client-internal nodes. The implications of this architectural decision are illustrated in Figures 17 and 18. The former presents the deployment view of the System under the chosen topology. The latter illustrates the relevant segment of the information view, showing how a Computation Task Specification is propagated through the tree from one Client to another. The tree's root corresponds to the System Operator's Service Node. Its immediate children represent the "central" Service Nodes within the organisations of Input, Output, and Computing Parties. These central nodes may have their own child nodes. For example, recalling the Data Custodian's scenario, a dedicated internal Service Node may be deployed. In practice, the depth of the tree may increase depending on the needs within an organisation.

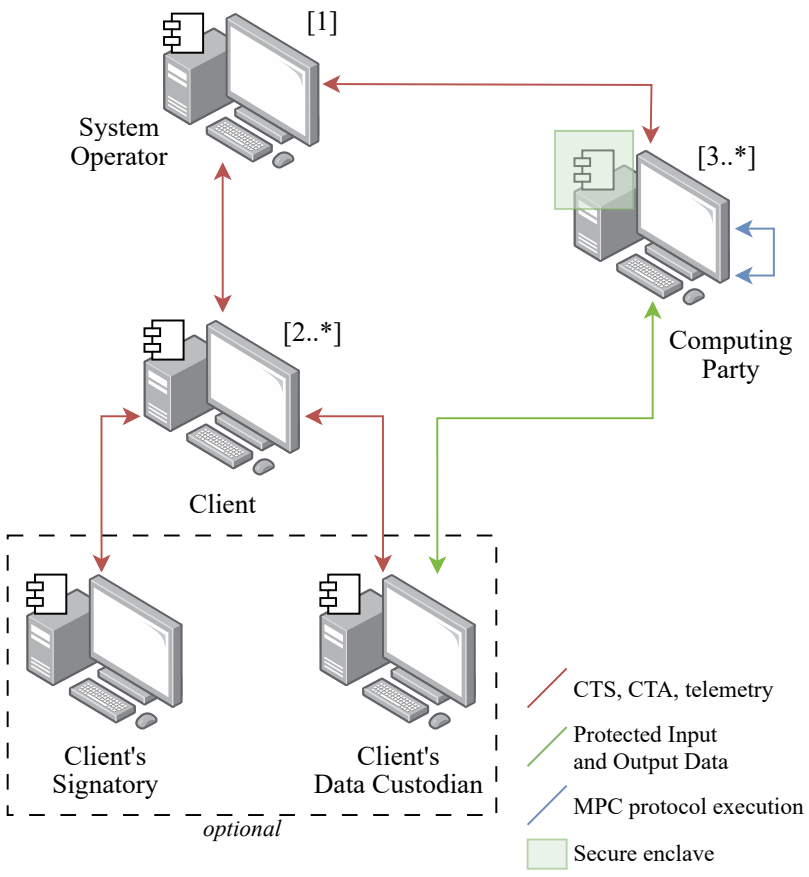


Figure 17. Deployment view of the System leveraging the tree topology of Service Nodes.

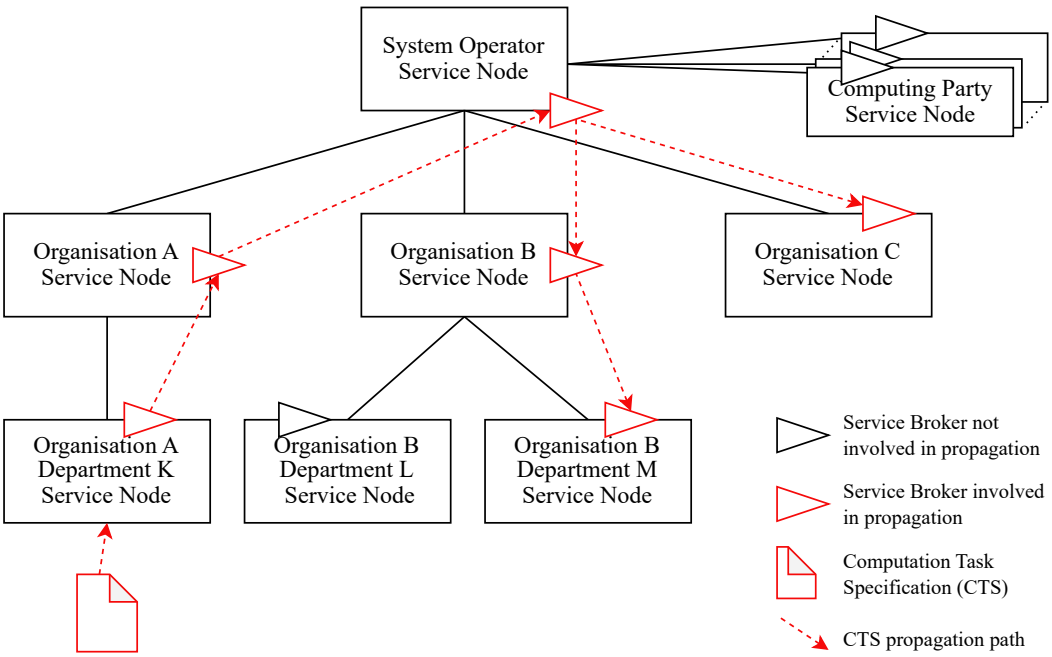


Figure 18. Propagation of a Computation Task Specification in a tree topology of Service Nodes.

This architectural decision has several consequences. First, it eliminates the need for a Data Custodian to trust intra-organisational internal infrastructure, which might otherwise represent a single point of trust. Second, it enables both Clients and the System Operator to define and enforce their policies regarding the child Service Nodes. Third, it simplifies management of the System by centralising governance of the Management Plane within the System Operator's Service Node. In line with the principle of plane separation, the Management Plane is logically and functionally isolated from the processing of Restricted Data. While the Control Plane and Data Plane components may influence computation and data transit, they are distributed across multiple Members, preventing a single point of trust. Nevertheless, this approach introduces a trade-off: it requires the development of a mechanism for propagating non-private data through the tree. For example, such artefacts are Computation Task Agreements and Computation Task Specifications that must be distributed across multiple parties to enable the functioning of the System. Appendix I (ADR-5) details this propagation mechanism.

4.2.6 Trust Establishment Between Parties

An additional important aspect of the System is establishing trust between the parties involved in a shared Computation Task. This is essential, as Peers must verify collaborators, Protected Output Data receivers and Computing Parties. This process should reduce the risks of participation in a Computation Task with a malicious or impersonated Input Party, Output Party or Computing Party. Furthermore, trust must be established independently from the System Operator.

To achieve this, the System employs certificates that are issued for each Service Node (refer to Appendix I, ADR-6). The design requires a hierarchical structure of certificate authorities (CA) responsible for issuing these certificates. This hierarchy aligns with the tree topology outlined in Section 4.2.5. The root certificate authority is represented by the System Operator, as its Service Node instance serves as the tree's root. Likewise, each "central" Service Node within an organisation can issue certificates for its departments or internal stakeholders, such as Data Custodian or Signatory. However, it is important to note that the described public key infrastructure is not designed to establish trust like the TLS/SSL protocols. The primary role of the root CA (i.e., the System Operator) is to distribute the certificates of Service Nodes participating in a Computation Task. Trust establishment, on the other hand, depends on out-of-band communication (for instance, via email) among the Peers, which is necessary not to introduce a single point of trust. This process consists of at least the following steps, as illustrated in Figure 19.

1. Obtain a certificate for the Service Node from within the System and generate a fingerprint.
2. Communicate with the local Task Organiser of the target Service Node through channels outside the System to retrieve the fingerprint of their certificate chain.
3. Compare the fingerprint obtained in step 2 with the fingerprint generated in step 1. Both fingerprints must match to establish trust towards the Service Node.

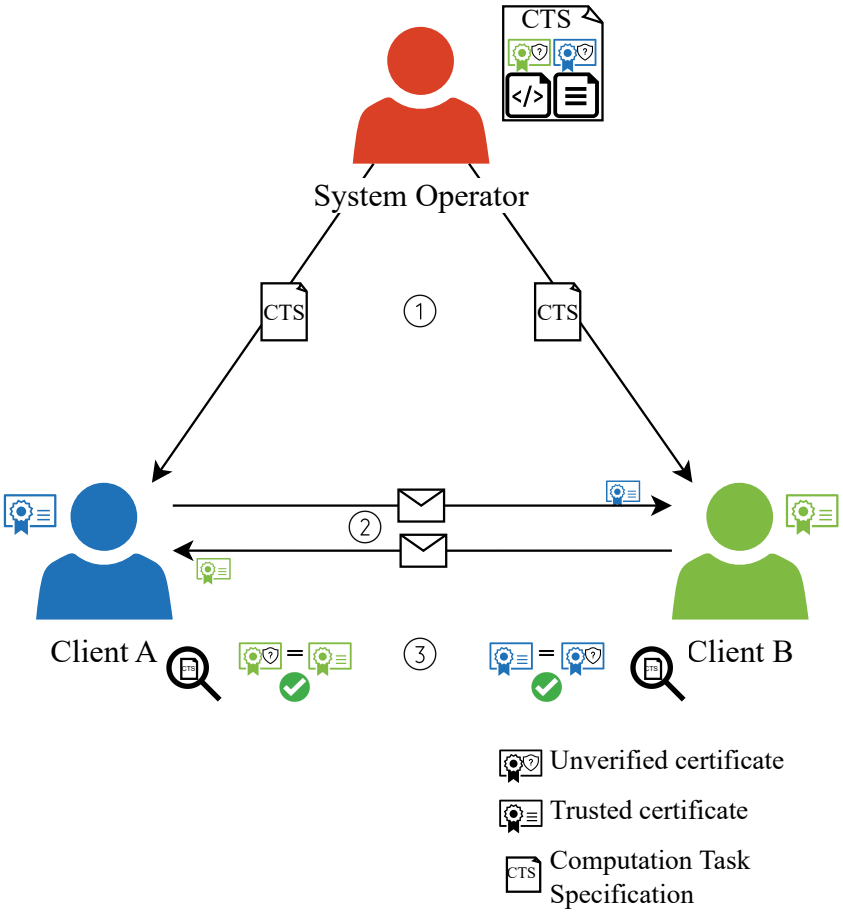


Figure 19. Trust establishment process between the Clients.

The Peers must also verify the identities of the Computing Parties, but not the other way round. Thus, out-of-band communication could be, for example, an email communication between the Clients and the organisation responsible for the respective Computing Party, or publishing certificates or fingerprints on the organisation's website.

The System requires each Service Node to maintain its own trusted view of other trusted Service Nodes. This approach allows the operator of a particular Service Node to determine which nodes can be trusted and which cannot. Trust in a Peer develops through prior collaboration in computations with said Service Node, as this implies that the procedure has been completed

beforehand, making subsequent or recurring collaboration easier and more secure. Architecture-wise, each Service Node is equipped with a local Identity Manager component. This component is primarily responsible for storing and propagating the certificates. Additionally, the trusted view of other Service Nodes can be linked to a set of permissions, thereby enhancing control over interactions with those Service Nodes.

4.2.7 Data Quality Assurance Algorithms

Another important aspect is the quality and trustworthiness of the submitted Input Data for a given Computation Task. Problems may arise from two distinct sources: low-quality data, which may result from accidental errors, and malicious data, which is submitted to manipulate the outcome or compromise privacy. Both can lead to misleading Output Data or the exposure of sensitive information. Therefore, the System should include mechanisms to identify such Input Data. This is also relevant in the context of avoiding a single point of trust, especially when a Computation Task involves two Input Parties, as each party implicitly trusts the other to provide accurate input.

The solution introduces data quality assurance algorithms (refer to Appendix I, ADR-7). These optional algorithms can be specified within the Computation Task Agreement. They are executed by all Computing Parties involved in the corresponding Computation Task. A quality assurance algorithm is essentially an MPC program that runs before the main Algorithm. It operates on the same Input Data, but produces a boolean Output Data indicating whether the submitted data meets predefined quality criteria. The main Algorithm is executed after valid Input Data has been successfully verified.

The primary purpose of these algorithms is to detect and filter out low-quality data. However, when carefully designed, they help identify malicious inputs that deviate from expected statistical patterns or logical constraints.

Considering consequences, this decision impacts the information view of the System in that the Computation Task Specification and Computation Task Agreement should include a field for specifying the quality assurance algorithms, using the same domain-specific language as the main Algorithm. The System must also support the execution of these algorithms prior to running the primary analysis. Nevertheless, Clients must be cautious when defining data quality criteria, as poorly designed validation procedures may inadvertently leak sensitive information during the validation process or the subsequent Algorithm execution.

4.2.8 Integrity Assurance for Computation Task Specification and Computing Node Software

As the Computation Task Specification and Computation Task Agreement are propagated across multiple Service Nodes, it is essential to implement measures that prevent tampering with these artefacts. Such tampering could include, for instance, falsifying Peers' identities or maliciously modifying MPC programs. However, other options exist to compromise the integrity of the Computation Task Specification or Computation Task Agreement.

To mitigate the risk of specifying a malicious Computing Node software, the System relies on publicly known digests of software images. Clients are required to specify the digest of the Computing Node software in the Computation Task Specification. Before Computation Task Agreement signing, the Clients must verify that the identifier of Computing Node software image specified in the Computation Task Specification matches the image that is indeed trusted and approved (refer to Appendix I, ADR-8).

Additionally, it is important to ensure the immutability of the Computation Task Specification after the corresponding Computation Task Agreement has been signed. To achieve this, each Client must sign a Computation Task Specification Addendum, which contains a hash of the Computation Task Specification being approved. This mechanism guarantees that tampering with the Computation Task Specification can be detected, as its integrity is cryptographically bound to the addendum.

This architectural decision affects the information view of the System by introducing the Computation Task Specification Addendum as a distinct artefact that must be signed. It also implies that all Peers must be aware of the digests corresponding to the Computing Node software images.

4.3 Built Attack Tree for JOCONDE System

In this section, we analyse the System by revisiting the attack tree presented in Section 4.1 and extending it with security controls of the System, as described in the Method (Section 3.3). These controls are designed to mitigate the previously identified threats. Furthermore, we append additional nodes to the attack tree to illustrate potential ways these security measures might be circumvented. The guidance for interpreting the diagrams remains unchanged from Section 4.1. However, we introduce one additional notation: green boxes, which represent the security controls implemented by the System to address a specific threat depicted in the parent

node. The child nodes following these control boxes indicate the conditions or steps required to bypass the corresponding security control.

Reuse Restricted Data without authorisation

To prevent unauthorised reuse of Restricted Data, the System implements three measures, as illustrated in Figure 20. These safeguards are designed to operate at different layers of the system architecture, addressing both hardware and software attack vectors.

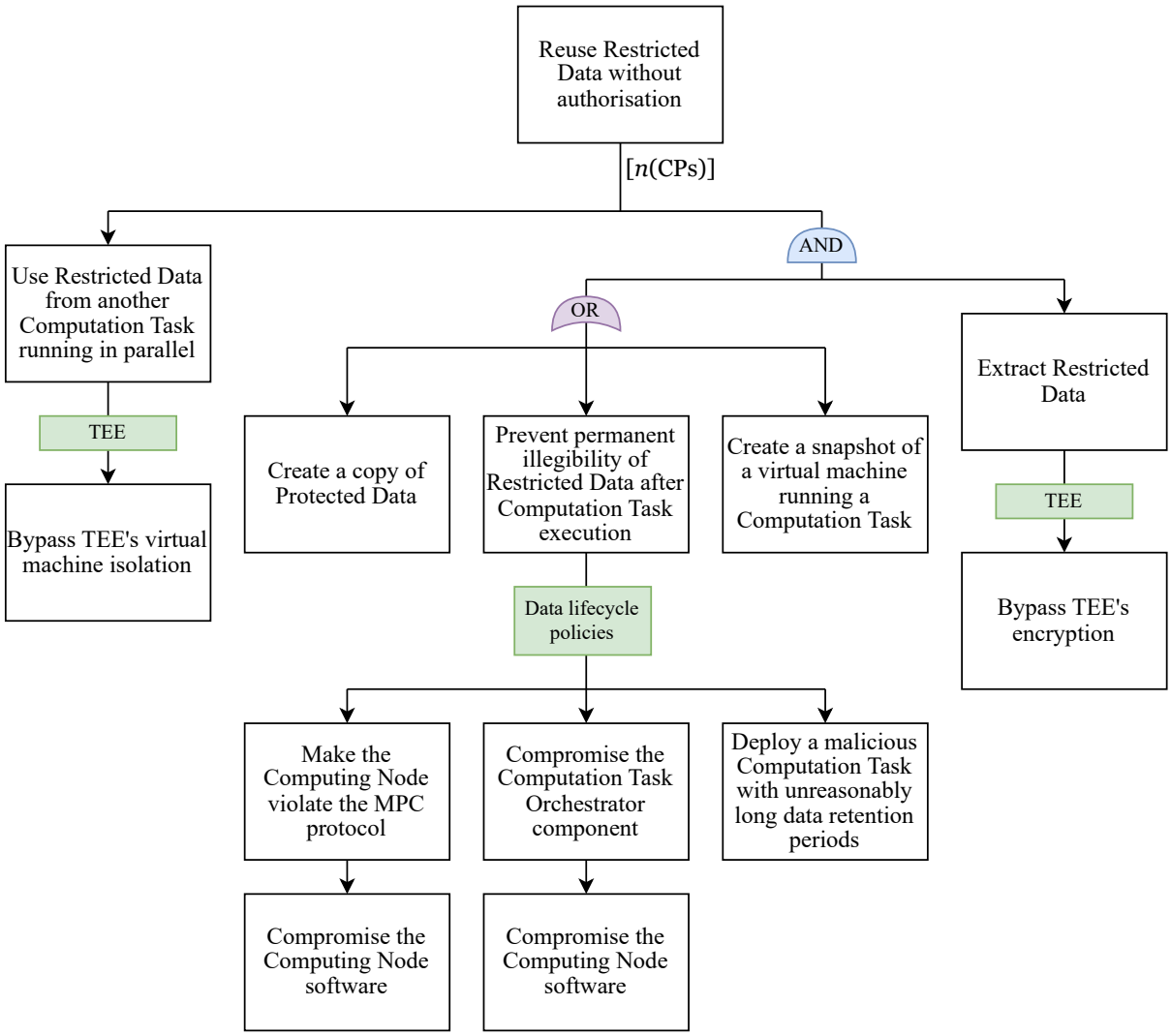


Figure 20. Attack tree for unauthorised Restricted Data reuse and the corresponding security controls.

First, the reuse of Restricted Data by any parallel Computation Task other than the authorised one is prevented through virtual machine isolation provided by the virtual-machine-based TEE. Second, the System enforces strict data lifecycle policies to ensure that Restricted Data is not retained beyond its required duration. Third, although it is theoretically possible to duplicate Protected Data or create a snapshot of the virtual machine executing the Computation Task, such

actions do not compromise the confidentiality of the Restricted Data, as it remains encrypted with a cryptographic key accessible only to the CPU within the TEE. Furthermore, for any of these attacks to succeed, the adversary must simultaneously achieve the same goal of the attack tree across multiple Computing Parties, with the required threshold determined by the security model of the underlying MPC engine, as discussed earlier.

Compromise confidentiality of Input Data

To prevent the interception of Protected Input Data, traditional information systems typically employ the TLS protocol. The JOCONDE System strengthens this approach by using a more secure variant – mutual TLS (mTLS) – as shown in Figure 21.

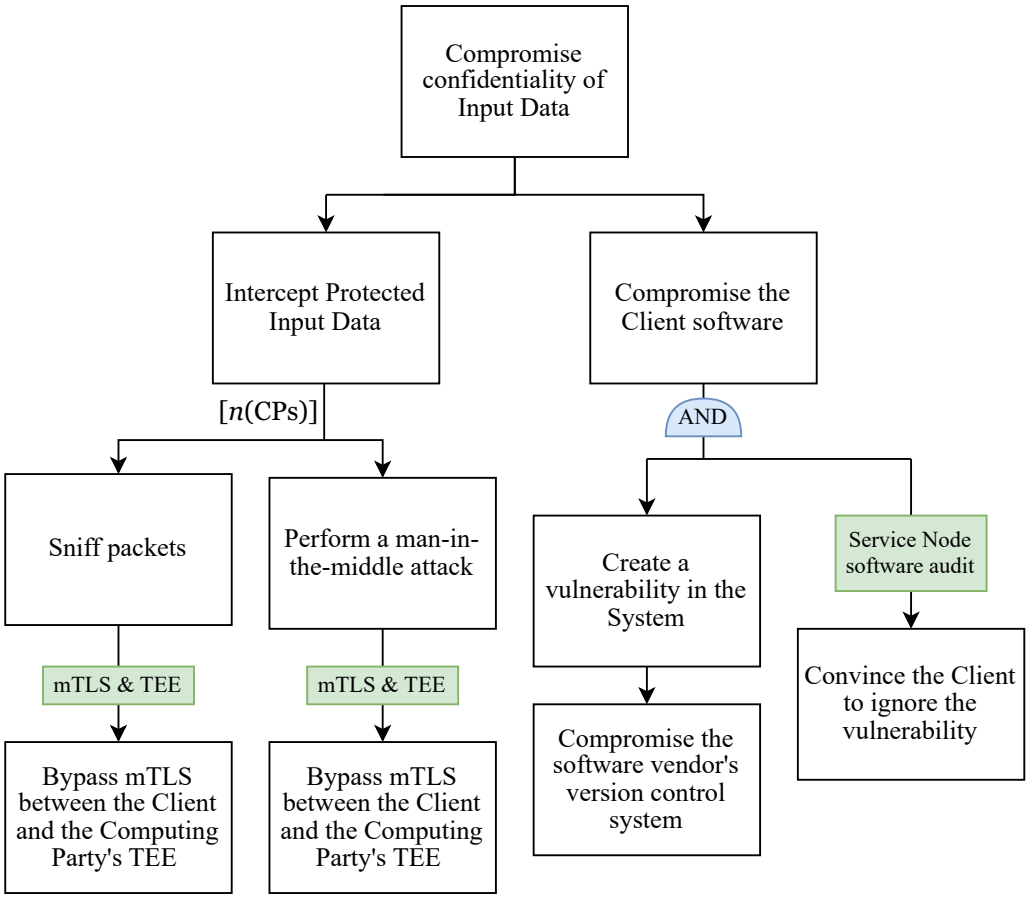


Figure 21. Attack tree for compromise of Input Data and the corresponding security controls.

This protocol protects communication against packet sniffing and man-in-the-middle attacks. Accordingly, an adversary would need to compromise the mTLS connections between the Input Parties and the TEEs of multiple Computing Parties involved in a computation to intercept Input Data. Regarding the compromise of Client software (specifically, the Service Node software in the context of the System), an independent audit is required prior to deployment. Given that

the Service Node software is open-source, a trusted auditor or reputable reviewer serves as an additional point of trust, thereby preventing the software vendor from becoming a SPoT-actor.

Compromise confidentiality of Output Data

Similar measures are applied to ensure the confidentiality of Output Data, as in the case of the compromise of Input Data, as illustrated in Figure 22. However, in the context of Output Data, impersonation poses an additional threat, requiring dedicated countermeasures.

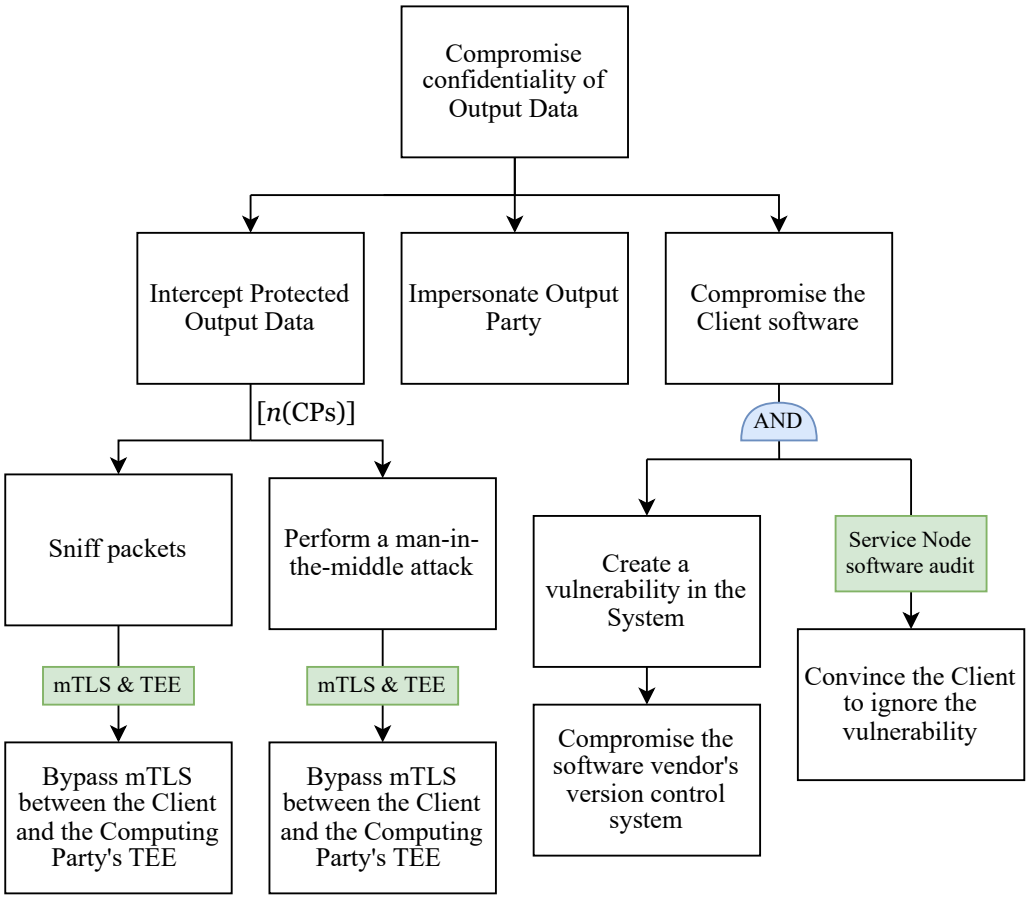


Figure 22. Attack tree for compromise of Output Data and the corresponding security controls.

To mitigate impersonation risks, the System employs a local Identity Manager component, certificate-based authentication, and out-of-band communication, as shown in Figure 23. Specifically, a successful impersonation attack requires stealing the Member's private key or tampering with their identity within a Computation Task Specification. The latter scenario entails gaining access to the System Operator's infrastructure, issuing a certificate signed by the parent Service Node, and compromising the Identity Manager component of each Peer. This component can be compromised in two ways. First, an attacker could gain physical or remote access to the Client's premises and compromise the System internally. Second, the attacker

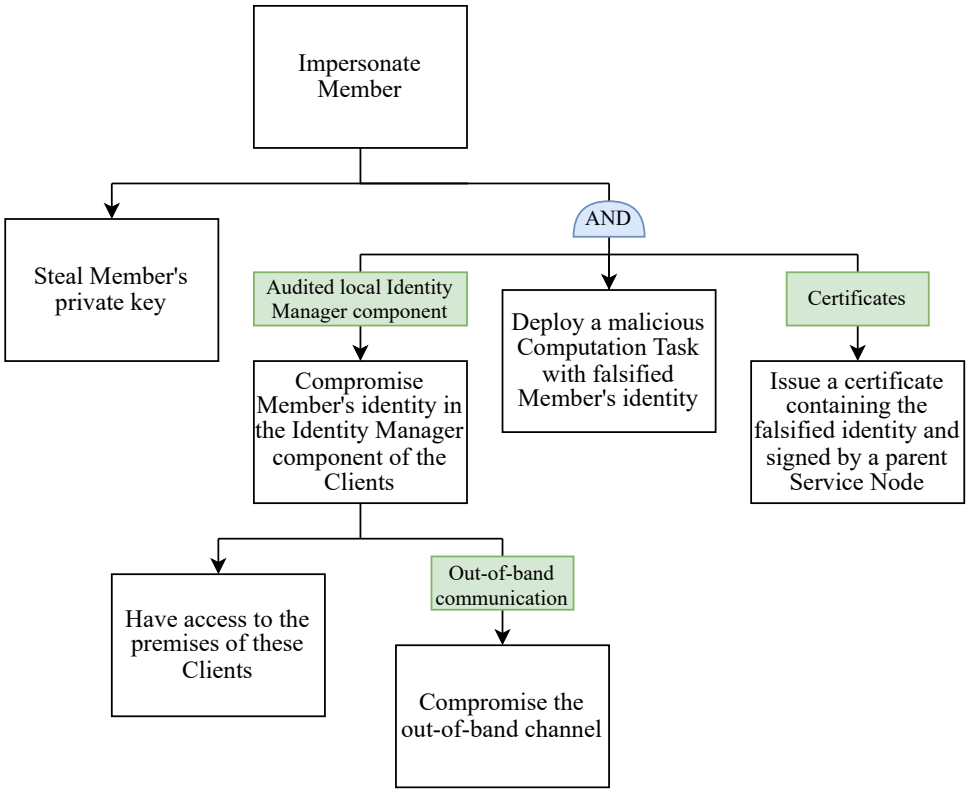


Figure 23. Attack tree for Member impersonation and the corresponding security controls.

might succeed by tampering with the out-of-band communication channel used for identity verification.

Reveal sensitive information through malicious Input Data submission

In the context of attacks involving malicious Input Data, the System applies a direct countermeasure, as illustrated in Figure 24. Specifically, each Computation Task may include data quality assurance algorithms defined by the Peers, as shown on the left side of the attack tree. As for the right side of the tree, impersonation has already been addressed previously. Threats of malicious Computation Task execution and compromise of the Computing Node software will be discussed in the following paragraphs.

Carry out a Computation Task on compromised Computing Nodes

An adversary may attempt to compromise the System by tampering with the Computing Node software, as illustrated in Figure 25. Two main attack vectors exist.

1. Introducing a vulnerability into the Computing Node software by compromising the software vendor and convincing Peers to disregard vulnerabilities.
2. Modifying the software after it has been deployed within the TEEs of Computing Parties.

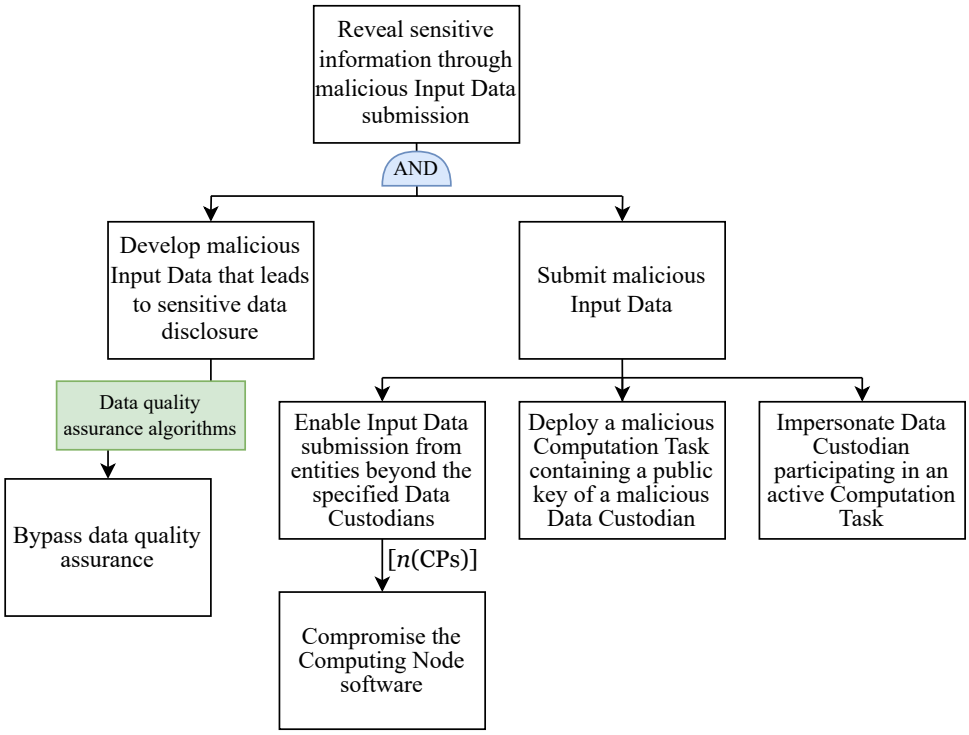


Figure 24. Attack tree for compromise of Restricted Data confidentiality through submission of malicious Input Data and the corresponding security controls.

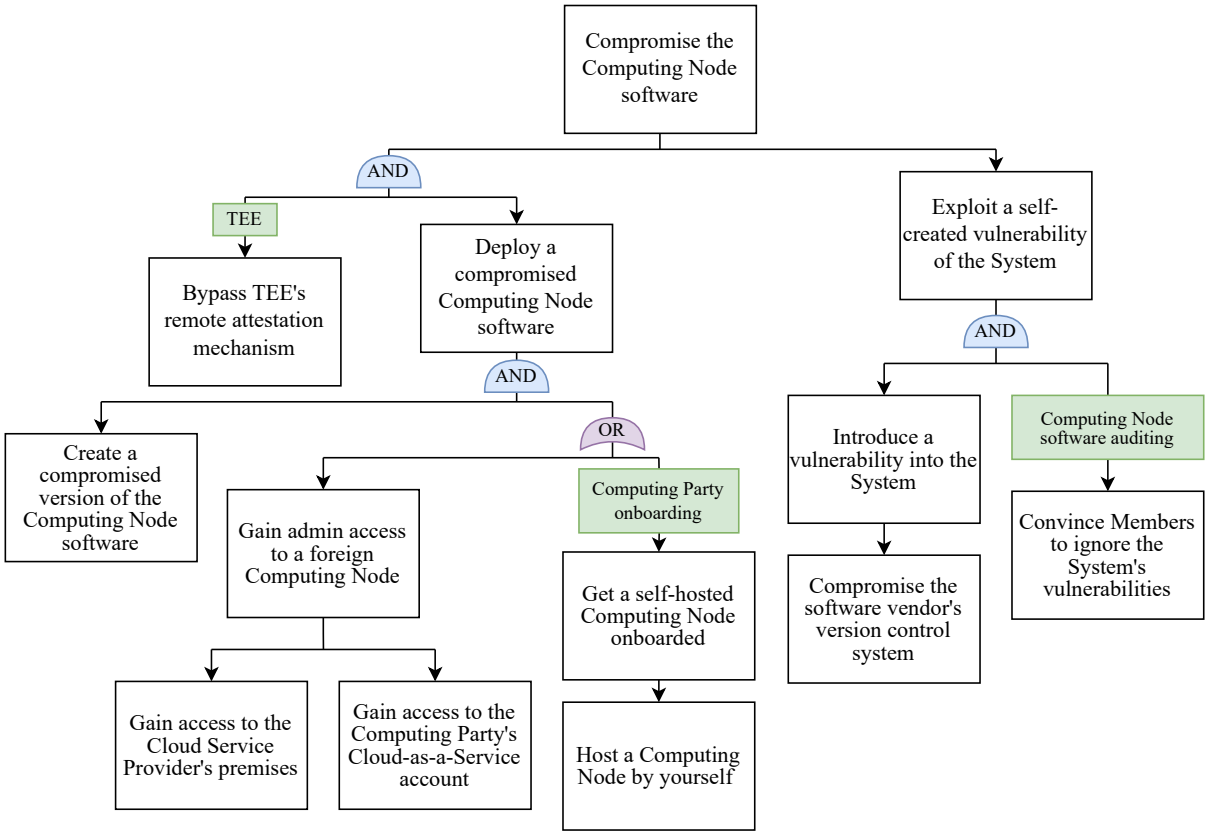


Figure 25. Attack tree for Computing Node software compromise and the security controls.

The System employs several mechanisms to minimise the risk of vulnerabilities in the Computing Node software and to ensure that the software cannot be tampered with during the execution of a Computation Task. Malicious modification at runtime is prevented through the remote attestation mechanism provided by the TEE. Moreover, the System is to mandate that highly reputable experts or independent auditors review the Computing Node software. This reduces the probability of some pre-existing vulnerabilities within the software. Nevertheless, even if an attacker succeeds in compromising the Computing Node software of a single Computing Party, more MPC nodes must still be compromised as well in order to violate the confidentiality guarantees, as discussed earlier.

Deploy a malicious Computation Task

Finally, although it is the responsibility of Peers to ensure that the Algorithm defined in a Computation Task Agreement does not reveal any confidential information, the System incorporates several safeguards to prevent the deployment of malicious Computation Tasks.

Three primary attack scenarios are considered. The first scenario involves the deployment of a forged Computation Task Specification. Since a Computation Task Specification must be approved before computation execution, an attacker must obtain valid signatures from all Peers, as presented in Figure 26.

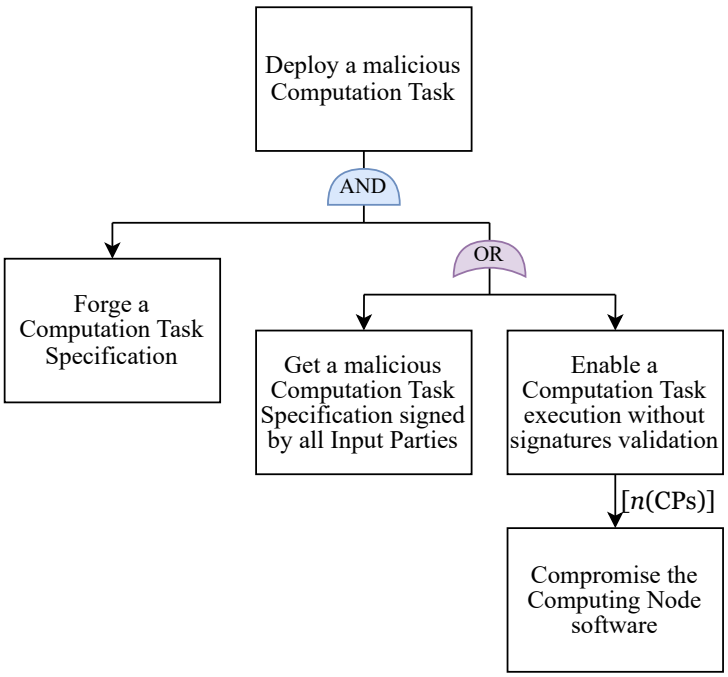


Figure 26. Attack tree for forging of a Computation Task Specification.

The dedicated attack tree for signing of a malicious Computation Task Specification is depicted in Figure 27[9]. One approach involves impersonating a Signatory by using the previously described strategy. Another involves deceiving an honest Signatory into approving a malicious Computation Task Specification by presenting it as legitimate. The latter entails convincing a reviewer to overlook vulnerabilities in the Algorithm, as well as persuading the Signatory to ignore inconsistencies in the identities of Peers, mismatched hashes in the core or addendum of the Computation Task Specification, and/or discrepancies in the digest of the MPC engine.

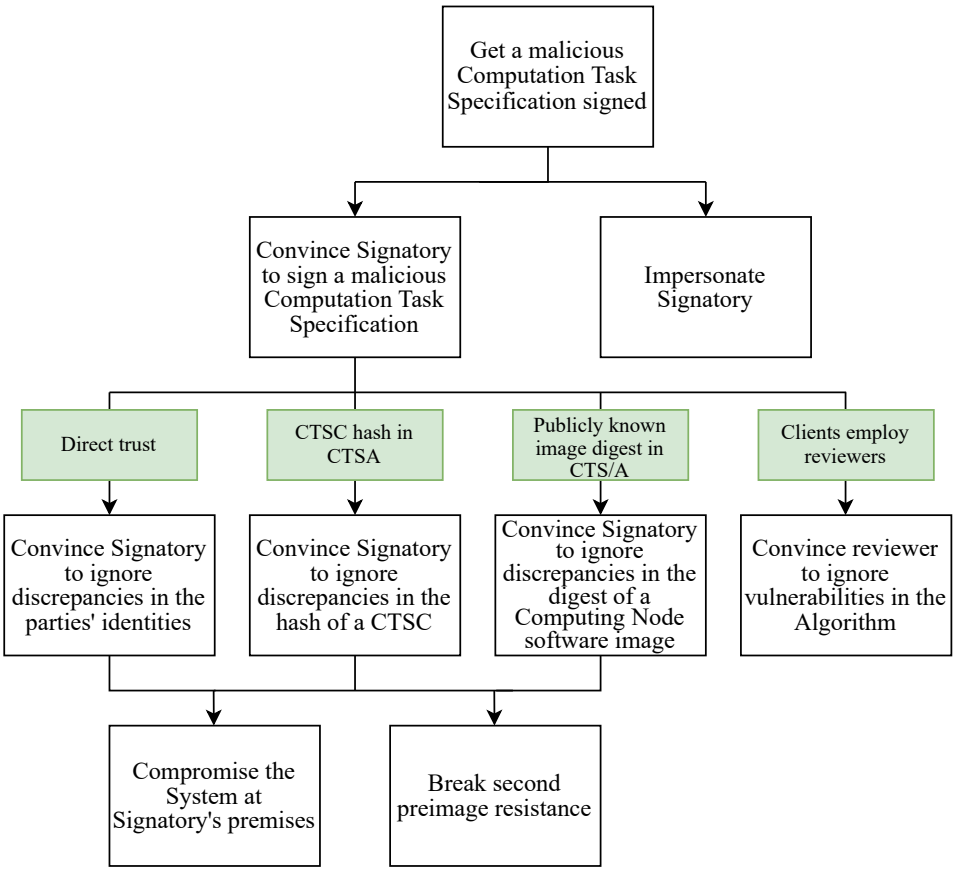


Figure 27. Attack tree for signing of a malicious Computation Task Specification and the corresponding security controls.

[9]We want to clarify two details about this attack tree. First, the third-level nodes converge into the exact fourth-level nodes, which is intentional. It aims to improve the readability of the attack tree. The tree demonstrates that convincing the Signatory to disregard discrepancies ultimately leads to one of two outcomes: compromising the System at the Signatory's premises or breaking second preimage resistance. Second, the tree represents the weakest attack of getting signed a malicious Computation Task Specification, where only one of its components is being tampered with. In practice, more complex attacks may require achieving multiple third-level goals concurrently, depending on which elements of the Computation Task Agreement the attacker intends to modify.

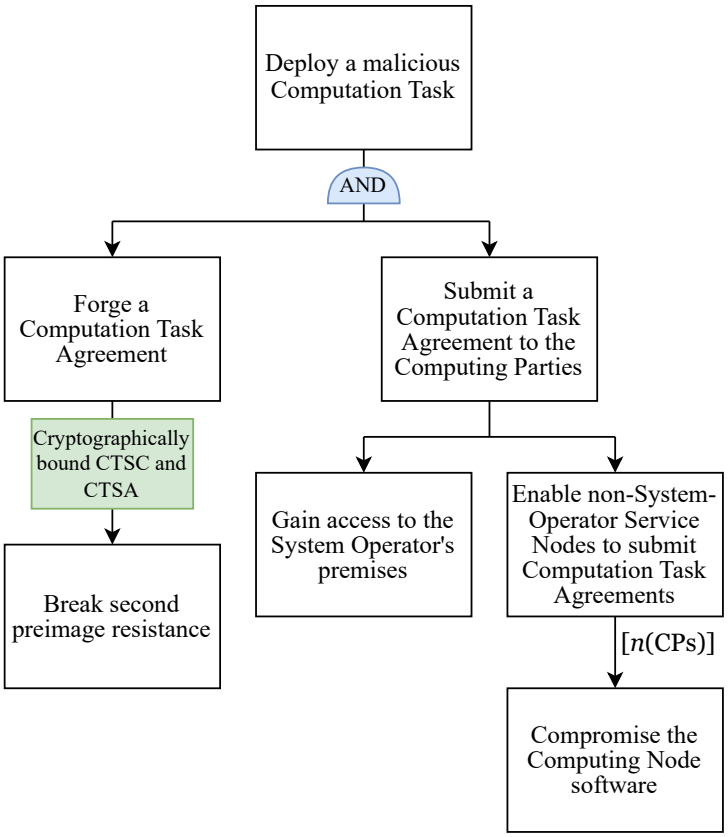


Figure 28. Attack tree for successful forging of a Computation Task Agreement and the corresponding security controls.

In the second scenario, an adversary must successfully forge a Computation Task Agreement, as presented in Figure 28. This requires breaking the cryptographic integrity of digital signatures. Specifically, crafting a malicious Computation Task Specification whose hash collides with that of a legitimate one.

Subsequently, the attacker must submit the forged Computation Task Agreement to the Computing Nodes, which is only possible via a Service Node operated by the System Operator. Hence, either access to the System Operator's infrastructure is required, or the Computing Node software must be tampered with to accept Computation Task Agreement submissions from unauthorised Service Nodes. As a third option (Figure 29), the attacker could submit a Computation Task Agreement with a mismatched hash; however, success in this case would necessitate compromising the remote attestation mechanism of the underlying TEE.

4.4 Summary

This chapter answered the research question RQ2: "What architectural decisions enable the design of the System with no single point of trust?". For that, we performed three steps. First,

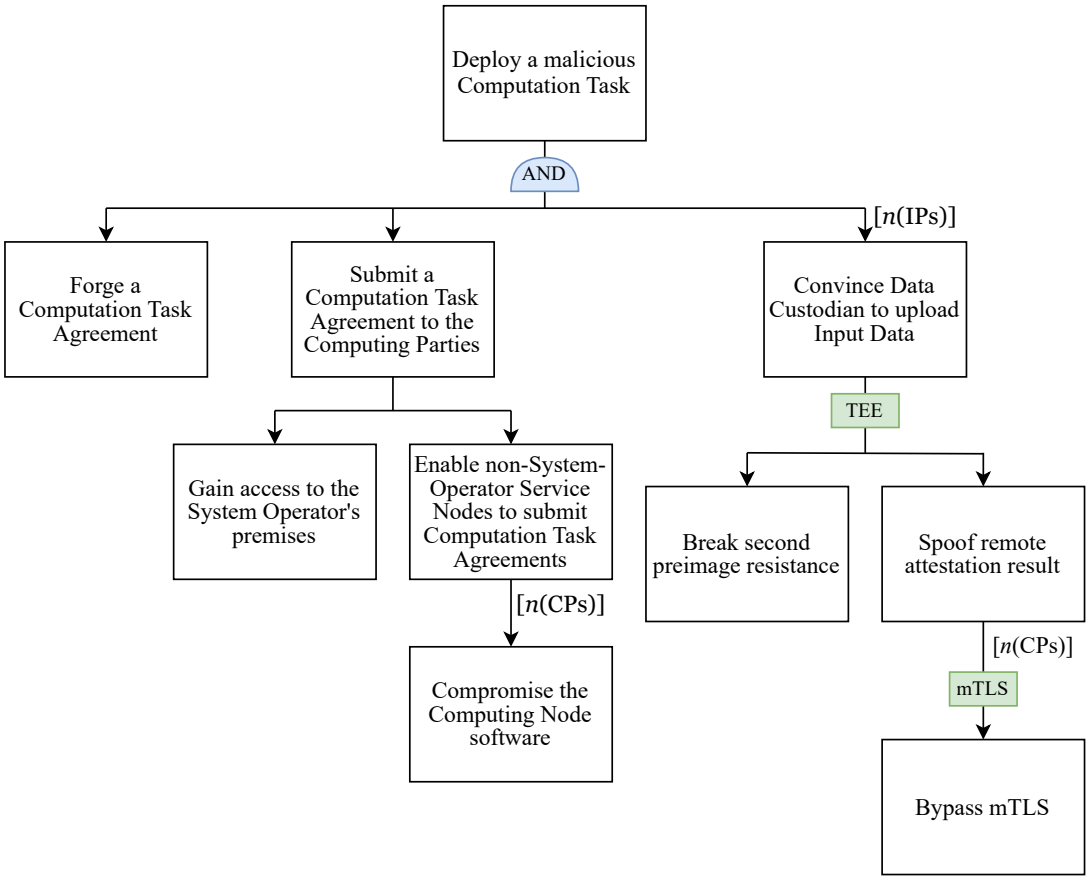


Figure 29. Attack tree for unsuccessful forging of a Computation Task Agreement and the corresponding security controls.

we introduced the attack tree constructed for the Minimum Viable MPC Framework to identify potential threats for a baseline architecture. Next, we formulated architectural decisions to mitigate these threats, considering the specifics of the System. Third, we updated the attack tree to reflect the impact of these architectural decisions. Our results revealed that the following architectural decisions enable the design of the System without introducing a single point of trust.

1. A minimum of three independent Computing Parties per Computation Task – to eliminate the need for a trusted third party in data analysis.
2. Direct communication between Clients and Computing Parties – to reduce reliance on intermediaries (e.g., reverse proxies) that may be controlled by a single entity, such as the System Operator.
3. Decentralisation of open-source or auditable services – to provide transparency and prevent the System Operator or the System vendor becoming a SPoT-like actor.

4. Combination of TEE and MPC – to integrate hardware- and cryptography-based trust anchors, thereby mitigating the risk of a single point of trust represented by a cloud service provider or a colluding subset of Computing Parties. The latter could occur if the collusions have not been identified during the selection of Computing Parties during the Computation Task negotiation despite their existence.
5. Tree topology – to avoid introducing intra-organisational single point of trust by ensuring that communication paths do not require trust towards the internal infrastructure of a given Peer.
6. Trust establishment mechanisms between Clients – to enable Clients to independently manage a trusted view of Peers' identities using out-of-band measures, without depending on the System Operator.
7. Use of data quality assurance algorithms – to detect data poisoning and prevent leakage of sensitive information.
8. Integrity assurance for the Computation Task Specification and Computing Node software – to ensure that Computation Task Agreements and the underlying software remain untampered and trusted throughout their execution.

Having identified SPoT-threats and formulated the respective architectural decisions, we proceed to the results evaluation.

5. Evaluation

This section addresses the research question RQ3: “Do the formulated architectural decisions achieve the System with no single point of trust?”. In order to answer the question, we validate the identified threats and formulated measures by interviewing three experts, as described in Section 3.4. Each interview validated one of the following properties.

1. **Completeness of threats** – to confirm that no threats related to a single point of trust are overlooked.
2. **Sufficiency of mitigations** – to evaluate whether the formulated architectural decisions mitigate the identified threats.
3. **Business feasibility** – to verify that the measures align with the business requirements of official statistics.

In the following sections, we present the results of the interviews and answer the corresponding research question.

5.1 Completeness of Threats

The first interview focused on assessing the completeness of the identified threats. We interviewed Riivo Talviste, an expert with 15 years of experience in developing and deploying secure MPC. His PhD thesis addresses the application of secure MPC in practice, and he has published several research papers on the topic.

The MPC expert pointed out minor issues, such as unclear wording in some attack tree nodes, and suggested restructuring particular fragments of the attack tree to enhance readability. He also noted that the number of Computing Parties that must be compromised to break an MPC system depends on the underlying security model of the selected engine. Indeed, MPC protocols differ in their security properties [14]. The models distinguish between passive, covert, and active regarding adversarial behaviour. Additionally, they vary based on the corruption threshold, i.e., whether a system assumes an honest or dishonest majority of Computing Parties. These parameters determine the maximum number of corrupted Computing Parties a system can tolerate before its security guarantees are violated. While such distinction is important when selecting or analysing specific MPC engines, it does not invalidate our threat model. By definition, no MPC engine should leak sensitive information when only a single Computing Party is compromised.

Accordingly, the threat scenarios considered in this work remain valid across different MPC security models.

Overall, the expert did not identify any overlooked or inadequately mitigated threats. He also validated that typical MPC-specific risks, such as collusion attacks and leakage during computation, were adequately considered. The interviewee found the identified threats realistic and relevant to deployment in the domain of official statistics.

5.2 Sufficiency of Mitigations

The second interview was conducted with Eduardo Ribas Brito, a security engineer and an expert in computer science and privacy-preserving technologies, with over five years of professional experience in the field. His research interests include federated data sharing, cryptography, and trustworthy software development. The interviewee is currently pursuing a PhD focused on decentralised infrastructure. He contributes to the European research projects and has authored several publications addressing trust, cybersecurity, and privacy-preserving technologies.

The interview focused on evaluating the sufficiency and effectiveness of the formulated mitigation measures. Several important observations emerged during the discussion.

First, the interviewee highlighted a potential single point of trust arising from the role of the System Operator when Peers are required to select the Computing Parties responsible for executing a Computation Task. This has been discussed in Section 4.2.6, and the solution for this potential single point of trust is the out-of-band communication between the Peers and Computing Parties.

Second, the expert commented on the implications of the tree topology of the Service Nodes. Although it does not handle confidential data directly, the System Operator and other Service Nodes may still represent a single point of trust in terms of availability. Specifically, any Service Node involved in the forwarding of Computation Task Specifications or Computation Task Agreements could launch a denial-of-service attack. While this thesis focuses on confidentiality, the availability issue falls outside the current scope and is considered a subject for future work.

Despite these concerns, the expert confirmed that the measures effectively mitigate the targeted threats. However, he noted that certain types of single point of trust remain unaddressed. For example, the expert referred to the risk of supply chain attacks. Such threats were acknowledged in Section 2, where we noted that some could not be resolved at the architectural level of the System. Indeed, a malicious or vulnerable third-party library used by an MPC engine

could introduce a single point of trust beyond the architectural scope. The broader issue of software trustworthiness is outside the focus of this thesis, though it is actively studied in other research [40]. Overall, the interviewee concluded that the measures are sufficient to mitigate the threats they intend to address within the defined scope of this study.

5.3 Business Feasibility

The third interview was conducted with Fabio Ricciato, project manager of the JOCONDE project on behalf of Eurostat. The interviewee holds a PhD in Information and Communication Technologies. He is currently working in the unit dealing with methodology and innovation in official statistics, with a focus on the role that privacy-preserving technologies may play for collaborative data analysis and cross-organisational statistical production.

This interview aimed to determine whether the formulated measures are feasible from a business perspective. During the interview, the project manager was presented with the practical implications of the architectural decisions. The discussion aimed to assess whether these implications align with the operational and regulatory needs of official statistics. For that, we focused on four categories of effects, and the feedback received is summarised below.

Infrastructure

The interview confirmed the feasibility of the decentralised architecture. One implication of this design choice is that both the System Operator and each Client are required to deploy and manage their own Service Nodes. The interviewee did not consider this problematic. However, it was emphasised that the System must be sufficiently flexible to accommodate deployment in diverse environments, such as public clouds, private clouds, or physical on-premises infrastructure. This level of flexibility is achievable without incurring significant additional effort.

Another implication of the measures is the onboarding responsibility placed on the System Operator for each new Client. Such responsibility was not considered a substantial concern, given that onboarding is a one-time process per Client. This is because the requirement for non-collusion among Computing Parties is not enforced during the onboarding. The requirement must instead be enforced at the stage of Computing Party selection for individual Computation Tasks.

Software/artefact verification

The evaluation further examined the business feasibility of requiring Clients to verify the correctness of Computation Task Specification, Computing Node software images, and the

identities and configurations of Computing Parties involved in a computation. The business feasibility of these processes depends on how effectively they are operationalised within the Service Node software. They are not expected to impose significant overhead on Clients if implemented efficiently. For instance, verifying short identifiers or configuration hashes was deemed manageable, whereas verification of longer, complex strings could be burdensome and must be avoided. Although some of these verification steps may require human involvement or out-of-band processes, the expert concluded that they would not introduce excessive overhead, assuming a reasonable level of operational support is implemented within the System.

The evaluation also addressed the System requirement for independent review or auditing of the Service Node software and Computing Node software. The interviewee did not identify this requirement as insurmountable. Theoretically, there are no limits to how deeply they may be reviewed for the open-source components of the Management Plane and the Control Plane. However, as being open-source alone does not assure trustworthiness, the expert finds that the System Operator should procure formal audits of these components.

In the case of the Data Plane, where proprietary technologies might be used, the only viable approach for integrating such components into the System is independent auditing by an appointed auditor. In other words, the owner of such components would need to provide access to and allow review of their source code. The expert considered potential supply chain attacks a general concern not unique to the System. No unique mitigation strategy is needed beyond standard industry practices, with the expectation that state-of-the-art security measures are implemented during the trust-building phase of the project.

Identities

During the interview, we also considered the implications of establishing direct out-of-band trust relationships between the Peers. The implications were assessed as acceptable because this is a one-time procedure per pair of Peers. First, Peers are assumed to be acquainted by the time a Computation Task negotiation is initiated. Second, the project currently targets a relatively small user base, a few tens of Users, making the effort required to establish direct trust relationships manageable. Similarly, adding trusted identities into the Identity Manager component was not viewed as problematic, assuming this operation is incorporated well into the user flow.

Furthermore, the tree topology model, where parent Service Nodes are responsible for signing the certificates of their child Service Nodes, was not identified as cumbersome for the Users. This process also requires incorporating out-of-band measures. As with other processes, the

expert emphasised the importance of operationalising the related procedures effectively to ensure a usable trust establishment workflow.

User interactions

The evaluation also addressed the allocation of responsibility for ensuring appropriateness of data quality assurance algorithms and the main Algorithm. Delegating this responsibility to the Clients is acceptable, provided that the System offers adequate support mechanisms. If certain types of malicious Input Data are known in advance, then corresponding quality assurance algorithms can be developed as standard components or libraries for each Computation Task. These may be either recommended for use or eventually integrated into the Computation Task definitions or even embedded within the MPC engines, particularly for commonly expected checks.

In the case of domain-specific malicious data, the System is expected to evolve as a learning platform, facilitating knowledge exchange among its Users. This includes recommending relevant domain-specific quality assurance algorithms and supporting the sharing of such practices across the user base. Unless there are valid reasons for restriction, the public availability of all Computation Task Specifications and Computation Task Agreements further supports collective learning and the continual improvement of quality assurance. These mechanisms complement the role of independent reviewers.

The inflexibility introduced by immutable Computation Task Agreements was not seen as problematic. Indeed, any modification in a Computation Task Specification would imply restarting the Computation Task negotiation process. This constraint aligns with the System's intended role as a last-resort mechanism, invoked only when conventional solutions are inadequate. While increased flexibility may be desirable in future iterations of the System, it is not considered a priority at the current stage of maturity.

Overall, it was recognised during the interview that the System introduces a novel architectural and operational paradigm, and its adoption naturally entails a degree of effort from all participants. Notably, the System is designed to serve as a last-resort solution, activated when standard approaches are not applicable. As such, it prioritises stronger trust guarantees at the expense of operational simplicity. The trade-off is deliberate: enhanced protection necessarily comes with increased procedural complexity. While some mechanisms may appear heavyweight or inconvenient, they align with the project's strategic objectives. From a risk management perspective, it is preferable to begin with stronger safeguards and gradually reduce them if

proven excessive, rather than attempt to scale up security post hoc in response to failures or reputational damage. Once the System matures and is better understood, opportunities may arise to simplify processes and improve business-friendliness, particularly if certain safeguards are empirically shown to be unnecessarily strict.

Summarising the interview results, the System is feasible to the extent realistically possible, given its unprecedented nature. It is acknowledged that feasibility does not imply seamless integration or minimal friction. Instead, the core requirements and responsibilities are achievable through reasonable operationalisation. The evaluation affirms the feasibility of the System within its defined scope and purpose.

5.4 Summary

This chapter presented the evaluation results. The evaluation consisted of three interviews with experts. Each interview focused on one of the three core properties: completeness of threats, sufficiency of mitigations, and business feasibility.

The first interview confirmed the validity of the threat model presented in this thesis. The second interview provided a review of the formulated mitigation measures. While the expert acknowledged that the measures are effective within the scope of the study, residual risks such as supply chain attacks and denial-of-service threats were also noted. These findings confirm that the architectural decisions are well-aligned with their intended purpose but should be complemented with additional considerations in future work. The third interview assessed the business feasibility of the architectural decisions. The feedback indicated overall alignment with the business requirements.

The results also enable us to answer the research question RQ3: “Do the formulated architectural decisions achieve the System with no single point of trust?”. Based on the expert evaluations, we conclude that the formulated architectural decisions eliminate all identified threats that could result in a single point of trust within the defined scope of this thesis. No critical single point of trust-related threats were overlooked. The measures are considered sufficient and do not contradict the business requirements of official statistics.

6. Conclusion

This thesis investigated how to design the JOCONDE System, which enables privacy-preserving cross-organisational data analysis, so that it does not introduce any single point of trust. By addressing this problem, the study formulated and evaluated multiple architectural decisions that ensure no single entity can compromise Users' sensitive data.

6.1 Limitations

Several limitations of this work should be acknowledged. The first limitation is that our study is purely theoretical. Even though the experts have confirmed the theoretical correctness of our results, further progress would benefit more from prototype validation. Without empirical implementation, it remains uncertain what challenges are to expect when realising the architectural decisions in practice. We are optimistic in this regard, as most decisions are based on the principles already adopted in other trust-critical systems. Second, the analysis assumes the correctness of software implementations. In practice, unforeseen implementation bugs or zero-day vulnerabilities could introduce SPoT-threats, even if the architecture is theoretically secure. Finally, the architectural decisions were explicitly designed to meet the requirements of official statistics. Applying these results in designing a more general MPCaaS system may require adaptation and reconsidering architectural trade-offs.

6.2 Answer to Research Question

The objective of this thesis was to formulate a set of architectural decisions that enable the design of the System with no single point of trust. The main research question was: "How to achieve no single point of trust for the System?". To address this question, we followed a method that consists of threat modelling, architectural analysis, and expert validation. The first step was to construct a threat model for a baseline architecture of an MPC system. This analysis, represented by the attack tree consisting of multiple fragments, revealed multiple SPoT-threats. Then, we formulated eight architectural decisions tailored to the specifics of the System to address these vulnerabilities. These decisions were based on ideas of decentralisation, direct trust, hardware- and cryptography-based security guarantees, and verifiability. Third, we revisited the initial threat model and expanded it with the specifics of the System and the formulated decisions. To validate these decisions, we conducted expert interviews focusing on three criteria: 1) completeness of the identified threats, 2) sufficiency of the formulated mitigations, and 3) compatibility with business requirements. Based on expert feedback, our

threat analysis was complete, the formulated architectural decisions are sufficient, and the measures are technically sound and aligned with the operational constraints of official statistics.

6.3 Future Work

Several directions for future research exist. First, investigating a concept of single point of trust concerning availability would be a step towards a more general MPCaaS system. Such research could address denial-of-service scenarios. These are relevant in settings where the assumption of honest Input Parties cannot be guaranteed – unlike the context of official statistics considered in the JOCONDE project. Second, further research could focus on the risks introduced by supply chain attacks. Specifically, it would be valuable to examine how trust assumptions might be undermined by compromising third-party components or services. This work would extend the current threat model to account for transitive trust vulnerabilities. Finally, developing and evaluating a prototype of the System would enable practical assessment of implementation feasibility, performance trade-offs, and the robustness of the architecture in real-world conditions. Accordingly, the next phase of the JOCONDE project involves prototype development based on our research findings.

List of References

[1] Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the Protection of Natural Persons with Regard to the Processing of Personal Data and on the Free Movement of Such Data, and Repealing Directive 95/46/EC (General Data Protection Regulation), OJ L 119, 4.5.2016, Pages 1–88. https://eur-lex.europa.eu/eli/reg/2016/679/oj. (03.10.2024).

[2] United Nations Economic Commission for Europe (UNECE). Project on Input Privacy Preservation: Final Report. https://statswiki.unece.org/download/attachments/395313305/UNECE%20IPP%20Final%20Report.pdf. 03/2023. (10.10.2024).

[3] Bogdanov D., Niitsoo M., Toft T., and Willemson J. High-Performance Secure Multi-Party Computation for Data Mining Applications. *International Journal of Information Security* 11.6 (11/2012), pp. 403–418. DOI: 10.1007/s10207-012-0177-2.

[4] European Commission. Call for Tenders ESTAT/2023/OP/0004. https://ec.europa.eu/info/funding-tenders/opportunities/portal/screen/opportunities/tender-details/docs/etender/12503/12503_158352_EN-ESTAT-2023-OP-0004-PETcall_TechSpecs_ENG_V1.pdf. 04/2023. (13.05.2025).

[5] Ricciato F. Steps Toward a Shared Infrastructure for Multi-Party Secure Private Computing in Official Statistics. *Journal of Official Statistics* 40.1 (03/2024), pp. 3–15. DOI: 10.1177/0282423X241235259.

[6] Eurostat CROS. JOCONDE. https://cros.ec.europa.eu/joconde. 02.05.2024. (03.10.2024).

[7] Hevner A. R., March S. T., Park J., and Ram S. Design Science in Information Systems Research. *MIS Quarterly* 28.1 (2004), p. 75. DOI: 10.2307/25148625.

[8] ADR GitHub Organization. Architecture Decision Records (ADR). https://adr.github.io/. (12.12.2024).

[9] International Organization for Standardization. ISO/IEC/IEEE 42010:2022(En), Software, Systems and Enterprise — Architecture Description. https://www.iso.org/obp/ui/#iso:std:iso-iec-ieee:42010:ed-2:v1:en. 2022. (12.02.2025).

[10] Tali K., Kisand A. D., Zakatov A., and Talviste R. JOCONDE D4.1. System Specification and Architecture. Tech. rep. 09.05.2025.

[11] Yao A. C. Protocols for Secure Computations. *23rd Annual Symposium on Foundations of Computer Science (Sfcs 1982)*. Chicago, IL, USA: IEEE, 11/1982, pp. 160–164. DOI: 10.1109/SFCS.1982.38.

[12] Shamir A. How to Share a Secret. *Communications of the ACM* 22.11 (11/1979), pp. 612–613. DOI: 10.1145/359168.359176.

[13] Cramer R. and Damgård I. Multiparty Computation, an Introduction. *Contemporary Cryptology*. Basel: Birkhäuser Basel, 2005, pp. 41–87. DOI: 10.1007/3-7643-7394-6_2.

[14] Pullonen-Raudvere P., Kisand A. D., Talviste R., and Tali K. JOCONDE D2.1. Technologies Analysis. Tech. rep. https://cros.ec.europa.eu/group/46/files/2500/download. 30.12.2024. (12.05.2025).

[15] Hastings M., Hemenway B., Noble D., and Zdancewic S. SoK: General Purpose Compilers for Secure Multi-Party Computation. *2019 IEEE Symposium on Security and Privacy (SP)*. San Francisco, CA, USA: IEEE, 05/2019, pp. 1220–1237. DOI: 10.1109/SP.2019.00028.

[16] Lindell Y. Secure Multiparty Computation. *Communications of the ACM* 64.1 (01/2021), pp. 86–96. DOI: 10.1145/3387108.

[17] Bogetoft P., Christensen D. L., Damgård I., Geisler M., Jakobsen T., Krøigaard M., Nielsen J. D., Nielsen J. B., Nielsen K., Pagter J., Schwartzbach M., and Toft T. Secure Multiparty Computation Goes Live. *Financial Cryptography and Data Security*. Ed. by Dingledine R. and Golle P. Vol. 5628. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 325–343. DOI: 10.1007/978-3-642-03549-4_20.

[18] Infocomm Media Development Authority. Digital Advertising in a Paradigm without 3rd Party Cookies. https://www.imda.gov.sg/-/media/imda/files/programme/pet-sandbox/imda-pet-sandbox--case-study--meta.pdf. 2023. (19.12.2024).

[19] Roseman Labs. Nederlandse Vereniging van Podotherapeuten (NVvP). https://rosemanlabs.com/en/customers/nederlandse-vereniging-van-podotherapeuten-nvvp. (11.10.2024).

[20] Maxwell N. Innovation and Discussion Paper: Case Studies of the Use of Privacy Preserving Analysis to Tackle Financial Crime. Future of Financial Intelligence Sharing (FFIS) Research Programme. https://www.gcffc.org/wp-content/uploads/2020/06/FFIS-Innovation-and-discussion-paper-Case-studies-of-the-use-of-privacy-preserving-analysis.pdf. 2020. (11.10.2024).

[21] Ballhausen H., Corradini S., Belka C., Bogdanov D., Boldrini L., Bono F., Goelz C., Landry G., Panza G., Parodi K., Talviste R., Tran H. E., Gambacorta M. A., and Marschner S. Privacy-Friendly Evaluation of Patient Data with Secure Multiparty Computation in a European Pilot Study. *npj Digital Medicine* 7.1 (10/2024), p. 280. DOI: 10.1038/s41746-024-01293-4.

[22] Willemson J. How Not to Use a Privacy-Preserving Computation Platform: Case Study of a Voting Application. *Computer Security*. Ed. by Katsikas S., Cuppens F., Cuppens N., Lambrinoudakis C., Kalloniatis C., Mylopoulos J., Antón A., Gritzalis S., Pallas F., Pohle J., Sasse A., Meng W., Furnell S., and Garcia-Alfaro J. Vol. 11980. Cham: Springer International Publishing, 2020, pp. 111–121. DOI: 10.1007/978-3-030-42048-2_8.

[23] Bogdanov D., Talviste R., and Willemson J. Deploying Secure Multi-Party Computation for Financial Data Analysis. https://eprint.iacr.org/2011/662. 2011. (04.10.2024).

[24] Talviste R. Applying Secure Multi-party Computation in Practice. http://hdl.handle.net/10062/50510. PhD thesis. University of Tartu, 2016. (15.11.2024).

[25] Cybernetica. Sharemind Developer Zone. https://docs.sharemind.cyber.ee/. (10.05.2025).

[26] Roseman Labs. Roseman Labs help center. https : / / support . rosemanlabs . com/. (10.05.2025).

[27] Talviste R., Tali K., Haav M., and Eerikson H. JOCONDE D1.1. Usage Scenarios and System Requirements. Tech. rep. https://cros.ec.europa.eu/group/46/files/2504/download. 20.01.2025. (12.05.2025).

[28] Partisia Blockchain Foundation. MPC Techniques Series, Part 10: MPC-as-a-Service – the Partisia Blockchain Infrastructure. https://medium.com/partisia-blockchain/mpc-techniques-series-part-10-mpc-as-a-service-the-partisia-blockchain-infrastructure-9b4833e77965. (04.05.2025).

[29] Kruchten P. The 4+1 View Model of Architecture. *IEEE Software* 12.6 (11/1995), pp. 42–50. DOI: 10.1109/52.469759.

[30] Rozanski N. and Woods E. Software Systems Architecture: Working with Stakeholders Using Viewpoints and Perspectives. 2nd ed., 3rd print. Upper Saddle River, NJ: Addison-Wesley, 2013.

[31] Shirey R. Internet Security Glossary, Version 2. Tech. rep. RFC4949. RFC Editor, 08/2007, RFC4949. DOI: 10.17487/rfc4949.

[32] Microsoft. The STRIDE Threat Model. https://learn.microsoft.com/en-us/previous-versions/commerce-server/ee823878(v=cs.20). 11/2009. (14.12.2024).

[33] MITRE. ATT&CK. https://attack.mitre.org/. (14.12.2024).

[34] Morana M. M. and Uceda Vélez T. Risk Centric Threat Modeling: Process for Attack Simulation and Threat Analysis. Hoboken, New Jersey: Wiley, 2015.

[35] Cybernetica. Sharemind MPC. https://cyber.ee/products/sharemind-mpc. (04.05.2025).

[36] Cosmian. CipherCompute. https://github.com/Cosmian/CipherCompute. (04.05.2025).

[37] Nygard M. Documenting Architecture Decisions. https://cognitect.com/blog/2011/11/15/documenting-architecture-decisions. 11/2011. (21.12.2024).

[38] Randmets J. and Kisand A. An Overview of Vulnerabilities and Mitigations of Intel SGX Applications. D-2-116. https://cyber.ee/uploads/D_2_116_An_Overview_of_Vulnerabilities_and_Mitigations_of_Intel_SGX_Applications_c1282b1505.pdf. 2024. (11.11.2024).

[39] Muñoz A., Ríos R., Román R., and López J. A Survey on the (in)Security of Trusted Execution Environments. *Computers & Security* 129 (06/2023), p. 103180. DOI: 10.1016/j.cose.2023.103180.

[40] Brito E., Castillo F., Pullonen-Raudvere P., and Werner S. TrustOps: Continuously Building Trustworthy Software. *Enterprise Design, Operations, and Computing. EDOC 2024 Workshops*. Ed. by Kaczmarek-Heß M., Rosenthal K., Suchánek M., Da Silva M. M., Proper H. A., and Schnellmann M. Vol. 537. Cham: Springer Nature Switzerland, 2025, pp. 53–67. DOI: 10.1007/978-3-031-79059-1_4.

Appendices

I. Architectural Decision Records

ADR-1 Minimum of Three Independent Computing Parties per Computation Task

Status

Accepted.

Context

According to MPC theory, a secure computation requires the involvement of at least two Computing Parties. In practice, however, it is common to involve a minimum of three Computing Parties per Computation Task [3]. To prevent breaches of Restricted Data confidentiality due to collusion, these parties must be completely independent [27].

Decision

This decision defines that each Computation Task be executed by no fewer than three independent Computing Parties. Ensuring independence of Computing Parties must be enforced at the parties' selection phase by the Peers.

Consequences

1. The System requires at least three independent Computing Parties to carry out a Computation Task, which differentiates the System from information systems utilising a client-server model. However, this is a standard practice for MPC systems.
2. The System enables selection of participating Computing Parties. This process must be performed carefully to prevent collusions of Computing Parties.
3. Users do not rely one single Computing Party (known as trusted third party) any more.
4. Computation remains secure provided that the number of compromised or colluding Computing Parties does not exceed the threshold tolerated by the used MPC engine.
5. The System requires a decision on how to enable Users to securely communicate with Computing Parties (refer to ADR-2).

ADR-2 Direct Communication Between Clients and Computing Parties

Status

Accepted.

Context

Each Computation Task execution requires transferring Input Data or Output Data between Clients and Computing Parties. As there are at least three Computing Parties involved in a Computation Task, one option is to run a reverse proxy server to simplify the process of Restricted Data transferring. However, such a server would be a single point of trust. Encryption of transmitted data could solve this problem, but nothing would prevent the host of the server from keeping the data transferred through this server. In this case, data would remain uncompromised as long as the used encryption algorithm is secure. However, encryption algorithms may contain vulnerabilities, enabling the breach of confidentiality of Restricted Data when exploits are known.

Decision

The System is to utilise direct communication between Clients and Computing Parties. The communication channels must be secure.

Consequences

1. Eliminates the need to trust a reverse proxy server.
2. Every Client must establish and maintain secure channels with multiple Computing Parties.
3. The System requires a decision on how to establish secure channels between Clients and Computing Parties (refer to ADR-4).
4. The need for secure and authenticated communication channels between all involved parties requires a key distribution and identity management mechanism.

ADR-3 Services Decentralisation

Status

Accepted.

Context

Among other functionalities, the System must enable Users to create Computation Tasks, sign Computation Task Agreements, upload Protected Input Data, and download Protected Output Data. In order to facilitate user interaction with the System, a front-end component accessible over the network is required. This component is called the *Client Portal* in the report "JOCONDE D1.1. Usage Scenarios and System Requirements" [27]. This component provides a user interface, but it also must interact with other components in a manner that can be trusted by Clients. The design of this component must avoid introducing a single point of trust and preserve the overall usability of the System.

The trade-off between centralised and decentralised services is closely tied to usability and trustworthiness. A centralised architecture, managed by the System Operator, would simplify the System development and maintenance. However, such an approach would make it difficult to build trust towards the System among Clients without reducing usability and maintainability.

Decision

The decision is to decentralise the System services, resulting in a network of multiple Service Nodes [10]. Each Client is expected to operate at least one Service Node hosting a subset of the System services. Similarly, the System Operator operates at least one Service Node to oversee and manage the System. The Control and the Management Plane components are open-source and to be audited. The review/audit is the prerequisite for Clients to trust the software.

Consequences

1. Facilitates trust establishment: Clients can deploy the open-source services on premises and trust them without additional overhead, although reviewing the source code before deployment remains necessary.
2. Decreases the number of measures required for trust establishment.
3. Increases development complexity: the System relies on multiple nodes that must communicate with each other to function correctly.

4. Shifts more responsibilities to the IT personnel of each Client. This can be partially mitigated by providing deployment scripts and support tools. Nonetheless, deploying the Service Node software is a one-time procedure.
5. The Control Plane and the Management Plane components must be audited/reviewed by independent and highly reputed entities.

ADR-4 Combination of TEE and MPC

Status

Accepted.

Context

Among the identified threats, several require the compromise of Computing Nodes. This can occur by compromising the Computing Node software. For instance, a compromised Computing Node software could allow an attacker to execute a Computation Task without validating signatures or to violate data lifecycle policies, potentially resulting in a breach of Restricted Data confidentiality. Therefore, the System must ensure the confidentiality of Restricted Data not only during transportation but also throughout and after computation.

Decision

The System is to utilise a virtual-machine-based trusted execution environment (TEE) with hardware-based encryption. In order to mitigate the risk of TEE vulnerabilities, the System is to use TEEs from different vendors. The Data Plane components are to be open-source or auditable. The review/audit is the prerequisite for Clients to trust the software.

Consequences

1. The Computing Node software is protected against compromise.
2. Restricted Data is safeguarded against tampering and eavesdropping during transmission. TEEs establish secure communication channels for both the upload and download of Restricted Data.
3. Restricted Data is protected against tampering and eavesdropping during and after computation, since it is encrypted with the encryption key bound to a virtual machine and accessible only to the CPU.
4. Restricted Data associated with different Computation Tasks is isolated to ensure data separation.
5. The System enables Users to remotely attest the software running within a TEE.
6. Additional software components (e.g., adapters) are required to support the use of various TEE technologies (e.g., virtual machine provisioning), as depicted in Figure 30.
7. Computation remains secure as long as the number of compromised TEEs does not exceed the threshold tolerated by the employed MPC engine.

8. The use of multiple TEE vendors mitigates the risk of any single vendor becoming a single point of trust.
9. The System provides the pre-baked Computing Node software images for virtual machine provisioning. The images are stored in the Image Repository component, as depicted in Figure 30.
10. The Data Plane and components must be audited/reviewed by independent and highly reputed entities before interfacing with the System.

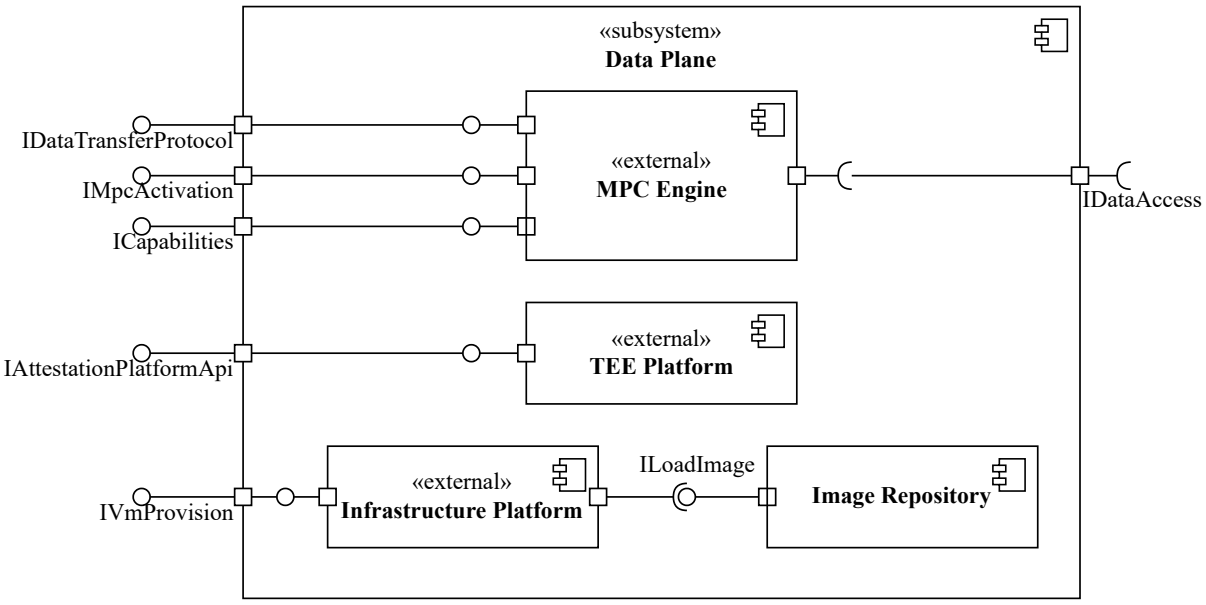


Figure 30. Functional view of the Data Plane [10:44].

ADR-5 Tree Topology

Status

Accepted.

Context

The System must support usability across organisations of different sizes and internal structures. Thereby, the System accommodates multiple Client-internal stakeholders, such as Data Analyst, Data Custodian, and Signatory. For example, the Signatory may be responsible for signing the Computation Task Agreement. In contrast, the Data Custodian typically handles the uploading of Protected Input Data. However, in some organisations, particularly those with strict internal data protection policies, Data Custodians are not permitted to disclose Restricted Data (associated with a Computation Task) to the organisation's broader infrastructure.

To respect such constraints, the System must enable Data Custodians to apply protection to Input Data locally and upload the resulting Protected Input Data directly to the Computing Nodes, thereby avoiding exposure of sensitive data within the organisational infrastructure. The same principle applies to the download of Protected Output Data and subsequent protection removal. While the decentralisation of the System infrastructure has already been analysed, a similar mechanism is needed to support intra-organisational trust models and security requirements.

Decision

Theoretically, multiple options exist to prevent the intra-organisational transmission of Input or Output Data, as outlined below.

1. Develop a browser-based application incorporating data protection and Computation Task integrity verification components.
2. Implement a lightweight version of the Service Node software, containing only the data protection and Computation Task integrity verification components.
3. Enable each Client-internal stakeholder to operate their own full-featured self-hosted Service Node.

The first option is highly complex from a technological aspect. The second option is technically less demanding. However, it introduces maintenance overhead by requiring the development of multiple software modules. In contrast, the third option implies no additional overhead, as the System is already designed to comprise multiple Service Nodes, as discussed in ADR-3. Furthermore, integration of any information system with the System necessitates an onboarding

process [27]. Regardless of the chosen approach, connecting a program or Service Node directly to the System Operator's Service Node instance would require both onboarding and certification for each new component or Service Node. This process is already established for full-fledged Service Nodes. Overall, the third option imposes the least operational and development burden, and thus it has been selected as the preferred solution.

The decision is to adopt a tree topology of Service Nodes to avoid the overhead associated with onboarding Client-internal nodes. The implications of this architectural decision are illustrated in Figures 17 and 18. The former presents the deployment view of the System under the chosen topology. The latter illustrates the relevant segment of the information view, showing how a Computation Task Specification is propagated through the tree from one Client to another. The tree's root corresponds to the System Operator's Service Node. Its immediate children represent the "central" Service Nodes within the organisations of Input, Output, and Computing Parties. These central nodes may have their own child Service Nodes. For example, recalling the Data Custodian's scenario, a dedicated internal Service Node software may be deployed. In practice, the depth of the tree may increase depending on the internal structure and requirements within an organisation.

Such a design decision necessitates a mechanism for propagating non-private data among the tree nodes. A dedicated component, called the *Service Broker*, performs this function. One example of such propagation is the synchronisation of the Computation Task Specification among all relevant parties. This scenario is illustrated in Figure 31. As shown in the figure, each Service Node includes a Service Broker; however, only the parties specified in the Computation Task Specification participate in its propagation. The corresponding Computation Task Agreement would also be propagated to the same parties, as well as to the relevant Computing Nodes, but this occurs downstream from the System Operator's node.

Consequences

1. Eliminates the need to trust intra-organisational infrastructure or that of the System Operator.
2. The System can be adopted by organisations regardless of their internal structure.
3. The System Operator is not required to perform onboarding or certification processes for Client-internal Service Nodes.

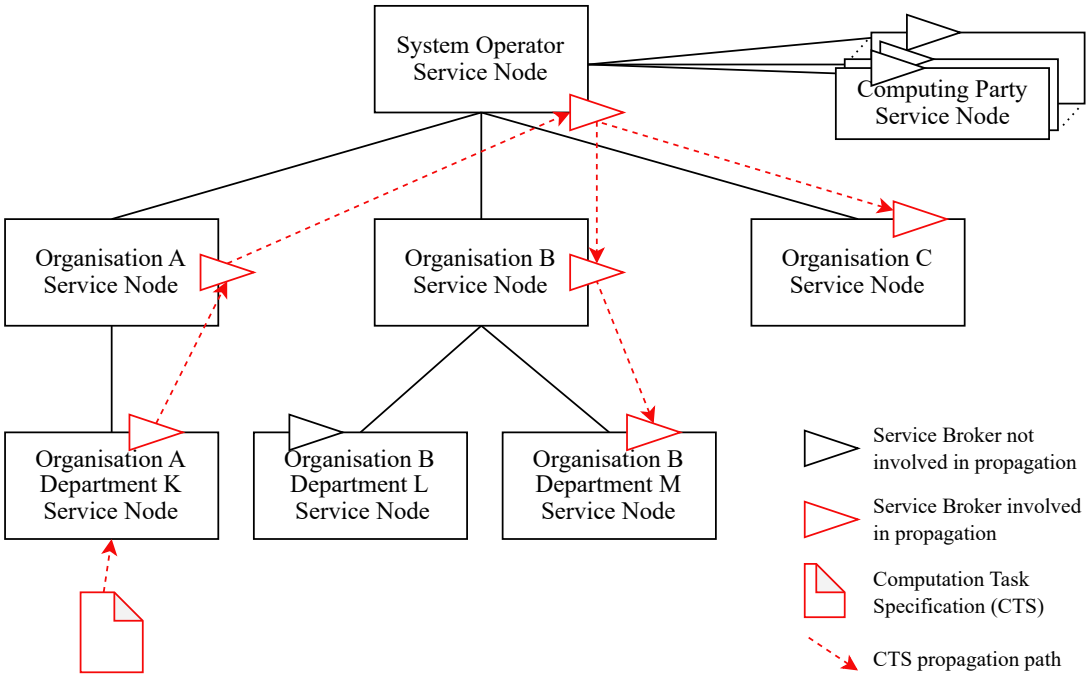


Figure 31. Propagation of a Computation Task Specification within a tree topology of Service Nodes.

4. Both Clients and the System Operator can define and enforce their own policies regarding subordinate Service Nodes.
5. Concerning the development view, there is no need to maintain specialised programming modules or supporting processes.
6. A routing mechanism must be provided to enable the Service Broker to forward messages correctly.
7. The decision increases the significance of machine-readable Computation Task Specifications and Computation Task Agreements.
8. Delegating infrastructure responsibilities to the Data Custodian reduces the operational burden associated with deploying a Service Node. However, this is a one-time procedure.

ADR-6 Trust Establishment Between Parties

Status

Accepted.

Context

An additional important aspect of the System is establishing trust between the parties involved in a shared Computation Task. This is essential, as Peers must verify collaborators, Protected Output Data receivers and Computing Parties. Failure to verify these entities may lead to participation in a Computation Task with a malicious or impersonated Input Party, Output Party or Computing Party. Furthermore, trust must be established independently from the System Operator.

Decision

To achieve this, the System employs certificates issued for each Service Node. The design requires a hierarchical structure of certificate authorities (CA) responsible for issuing these certificates rooted at the System Operator. This hierarchy aligns with the tree topology outlined in ADR-5. The actual trust verification occurs via out-of-band communication between the Peers (for instance, via email). Local certificate authorities within organisations of Peers issue certificates to the downstream entities (e.g., departments or stakeholders). However, it is important to note that the public key infrastructure is not designed to establish trust like the TLS/SSL protocols. The primary role of the root CA (i.e., the System Operator) is to distribute the certificates of Service Nodes participating in a Computation Task.

The procedure for trust establishment involves (as depicted in Figure 32):

1. Obtain a certificate for the Service Node from within the System and generate a fingerprint.
2. Communicate with the local Task Organiser of the target Service Node through channels outside the System to retrieve the fingerprint of their certificate chain.
3. Compare the fingerprint obtained in step 2 with the fingerprint generated in step 1. Both fingerprints must match to establish trust towards the Service Node.

The Peers must also verify the identities of the Computing Parties, but not the other way round. Thus, out-of-band communication could be, for example, an email communication between the Clients and the organisation responsible for the respective Computing Party, or publishing certificates or fingerprints on the organisation’s website.

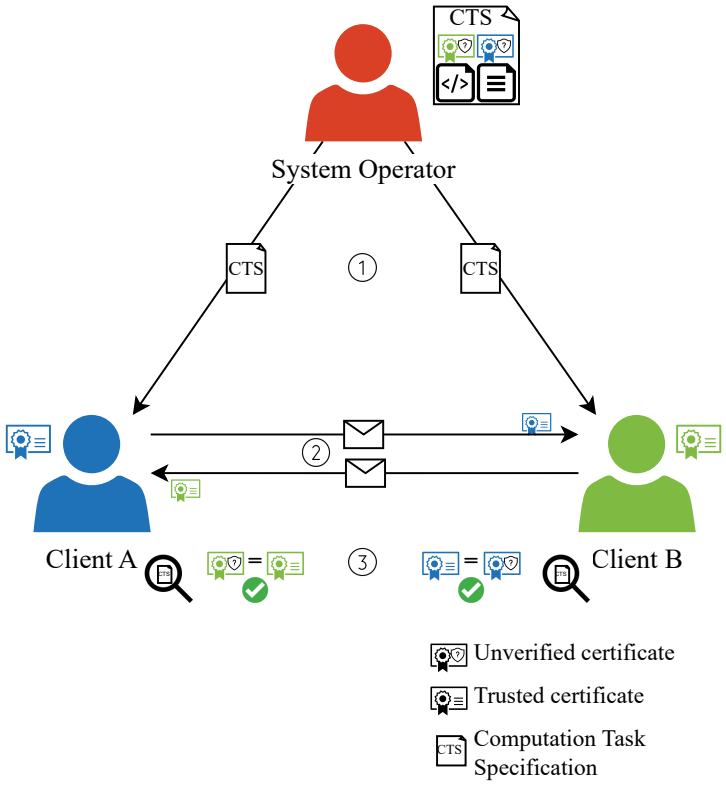


Figure 32. Figure depicting trust establishment between the Clients without relying solely on the System Operator.

The System requires each Service Node to maintain its own trusted view of other trusted Service Nodes. Architecture-wise, each Service Node is equipped with a local Identity Manager component. This component is primarily responsible for storing and propagating the certificates. Additionally, the trusted view of other Service Nodes can be linked to a set of permissions, thereby enhancing control over interactions with those Service Nodes.

Consequences

1. Decentralised trust verification eliminates the risk of a single point of trust, enhancing the System's security posture.
2. The requirement for out-of-band communication introduces additional coordination overhead, particularly during the initial trust establishment phase. However, the overhead is negligible as it is expected that Peers discuss the details of a Computation Task via an out-of-band channel as well.
3. This decision introduces a new component known as *Identity Manager*, responsible for storing and propagating the certificates.

ADR-7 Data Quality Assurance Algorithms

Status

Accepted.

Context

Another important aspect is the quality and trustworthiness of the submitted Input Data for a given Computation Task. Problems may arise from two distinct sources: low-quality data, which may result from accidental errors, and malicious data, which is submitted to manipulate the outcome or compromise privacy. Both can lead to misleading Output Data or the exposure of sensitive information. Therefore, the System should include mechanisms to identify such Input Data. This is also relevant in the context of avoiding a single point of trust, especially when a Computation Task involves two Input Parties, as each party implicitly trusts the other to provide accurate input.

Decision

The solution involves introducing data quality assurance algorithms. These optional algorithms can be specified within the Computation Task Agreement. They are executed by all Computing Parties involved in the corresponding Computation Task. A quality assurance algorithm is essentially an MPC program that runs before the main Algorithm. It operates on the same Input Data, but produces a boolean Output Data indicating whether the submitted data meets predefined quality criteria. The main Algorithm is executed after valid Input Data has been successfully verified.

The primary purpose of these algorithms is to detect and filter out low-quality data. However, when carefully designed, they help identify malicious inputs that deviate from expected statistical patterns or logical constraints.

Consequences

1. Computation Task Specification and Computation Task Agreement must include a field to specify the quality assurance algorithm, using the same domain-specific language as the Algorithm.
2. The System must implement functionality to execute quality assurance algorithms prior to the Algorithm.
3. Clients must carefully design quality assurance algorithms to avoid unintentional data leakage during validation or Algorithm execution.

ADR-8 Integrity Assurance for Computation Task Specification and Computing Node Software

Status

Accepted.

Context

As the Computation Task Specification and Computation Task Agreement are propagated across multiple Service Nodes, it is essential to implement measures that prevent tampering with these artefacts. Such tampering could include, for instance, falsifying Peers' identities or maliciously modifying MPC programs. However, other options exist to compromise the integrity of the Computation Task Specification or Computation Task Agreement.

Decision

To mitigate the risk of specifying a malicious Computing Node software, the System relies on publicly known digests of software images. Clients are required to specify the digest of the Computing Node software in the Computation Task Specification. Before Computation Task Agreement signing, the Clients must verify that the identifier of Computing Node software image specified in the Computation Task Specification matches the image that is indeed trusted and approved.

Additionally, it is important to ensure the immutability of the Computation Task Specification after the corresponding Computation Task Agreement has been signed. To achieve this, each Client must sign a Computation Task Specification Addendum, which contains a hash of the Computation Task Specification being approved. This mechanism guarantees that tampering with the Computation Task Specification can be detected, as its integrity is cryptographically bound to the addendum.

Consequences

1. The information view must be extended to include the Computation Task Specification Addendum as a signed artefact.
2. Clients are responsible for validating the digests of trusted Computing Node software images.
3. Any post-signature modification to the Computation Task Specification will invalidate the associated addendum, ensuring tampering is detectable.

II. Licence

Non-exclusive licence to reproduce the thesis and make the thesis public

I, **Anton Zakatov**,

(author's name)

1. grant the University of Tartu a free permit (non-exclusive licence) to

   reproduce, for the purpose of preservation, including for adding to the digital archives of the University of Tartu until the expiry of the term of copyright, my thesis

   Architecture of Secure Multi-Party Computation as a Service with No Single Point of Trust,

   (title of thesis)

   supervised by Kert Tali and Raimundas Matulevičius;

   (supervisors' names)

2. grant the University of Tartu a permit to make the thesis specified in point 1 available to the public via the web environment of the University of Tartu, including via the digital archives, under the Creative Commons licence CC BY NC ND 4.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright;

3. am aware of the fact that the author retains the rights specified in points 1 and 2;

4. confirm that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Anton Zakatov

15/05/2025

Acknowledgements

This thesis has been developed in close association with the JOCONDE project. Some results presented here have been incorporated into deliverables prepared by Cybernetica AS as part of a procured project by Eurostat (European Commission). 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.

The author wishes to thank his colleagues from Cybernetica for their ideas, support, and engagement throughout the work, as well as his supervisors for their guidance and constructive feedback.

Throughout the writing of this thesis, the following tools were used to enhance readability and ensure grammatical correctness: Grammarly[10] and ChatGPT (model 4)[11].

[10] Grammarly: https://app.grammarly.com/
[11] ChatGPT: https://chatgpt.com/