swarmchestrate

APPLICATION-LEVEL SWARM-BASED ORCHESTRATION ACROSS THE CLOUD-TO-EDGE CONTINUUM

# D3.1 Definition of Trust and initial design of the trust and privacy management systems

| DOCUMENT INFORMATION | |
|---|---|
| Work Package | WP3 |
| Task | T3.1, T3.2, T3.3, T3.4, T3.5 |
| Due Date | 30.6.2025 |
| Submission Date | 30.6.2025 |
| Lead Partner | UL |
| Deliverable Type | Report |
| Dissemination Level | Public |
| Version | 1.0 |
| Author(s) | Petar Kochovski, Vlado Stankovski, Giorgos Lagos, Fotis Paraskevopoulos, George Georgakakos, Yiannis Verginadis, Reyhaneh Rabbaninejad |
| Reviewer(s) | Amjad Ullah, Amy L. Murphy |

Co-funded by the European Union

UK Research and Innovation

## DOCUMENT HISTORY

| Version | Date | Contributor(s) | Short Description |
|---|---|---|---|
| 0.1 | 05.02.2025 | Petar Kochovski | Initial draft |
| 0.2 | 31.03.2025 | Petar Kochovski | Definition of trust and trust attributes |
| 0.3 | 09.04.2025 | Petar Kochovski, Vlado Stankovski | Trust management system |
| 0.4 | 15.05.2025 | Giorgos Lagos, Fotis Paraskevopoulos | Identity and Role management system |
| 0.5 | 23.05.2025 | George Georgakakos, Yiannis Verginadis | Populating definitions and respective information for Decentralized KB |
| 0.6 | 09.06.2025 | Reyhaneh Rabbaninejad | Private Resource Ranking in Swarmchestrate |
| 0.7 | 10.06.2025 | Petar Kochovski, Giorgos Lagos, Reyhaneh Rabbaninejad, Yiannis Verginadis | Swarmchestrate Knowledge Management Layer |
| 0.8 | 16.6.2025 | Amjad Ullah, Amy L. Murphy | Internal review |
| 0.9 | 25.6.2025 | Petar Kochovski, Vlado Stankovski | Addressing internal review comments |
| 1.0 | 30.6.2025 | Petar Kochovski | Final version |

## DISCLAIMER

This document has been prepared by Swarmchestrate project partners as a result of work carried out within the Grant Agreement contract Nº 101135012. Neither the Project Coordinator, nor any signatory party of the Swarmchestrate Project Consortium Agreement, nor any person acting on behalf of any of them makes any warranty or representation whatsoever, expressed or implied:

- with respect to the use of any information, apparatus, method, process, or similar item disclosed in this document, including merchantability and fitness for a particular purpose; or
- that such use does not infringe on or interfere with privately owned rights, including any party's intellectual property; or
- that this document is suitable to any particular user's circumstance; or
- assumes responsibility for any damages or other liability whatsoever (including any consequential damages) resulting from your selection to use this document or any information, apparatus, method, process, or similar item disclosed in this document.

The project is funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union. Neither the European Union nor the granting authority can be held responsible for them.

This work was co-funded by UK Research and Innovation (UKRI) under the UK government's Horizon Europe funding guarantee (grant numbers 10101169 and 10102651).

This work was supported by Seoul National University (Institute of Engineering Research) and through grant NRF-RS-2023-00302083 by the National Research Foundation of Korea (NRF), which is funded by the Korea government, Ministry of Science and ICT (MSIT).

## COPYRIGHT NOTICE

# Table of Contents

# List of Figures

# List of Tables



# Abbreviations

| Term | Meaning |
|---|---|
| AWS | Amazon web Services |
| DAO | Decentralized Autonomous Organization |
| DDH-f | Decisional Diffie-Hellman with an Easy Subgroup |
| DID | Decentralized Identifier |
| DLOG | Discrete Logarithm |
| EWMA | Exponentially Weighted Moving Average |
| FE | Functional Encryption |
| IPFE | Inner Product Functional Encryption |
| MCIPFE | Multi-Client Inner Product Functional Encryption |
| MPC | Multi-Party Computation |
| NIZK | Non-Interactive Zero-Knowledge |
| P2P | Peer-to-Peer |
| PPT | Probabilistic Polynomial Time |
| PRF | Pseudo-Random Function |
| RA | Resource Agent |

# Executive summary

Deliverable D3 .1 “Definition of Trust and Initial Design of the Trust- and Privacy-Management Systems” establishes the conceptual foundations and first architectural blueprint that will allow Swarmchestrate to make trustworthy, evidence-based, privacy-preserving orchestration decisions across the Cloud-to-Edge continuum. After analysing the state of the art and project requirements, the document formalises a trust model that is encompassing identity, behavioural and contextual dimensions, and specifies the attributes, data sources and timing aspects needed to operationalise that model within swarm environments. On top of this conceptual layer, four mutually-reinforcing systems are introduced: (i) a Trust-Management System that fuses various metrics into verifiable trust scores; (ii) an Identity- and Role-Management System that anchors decentralised identifiers (DIDs) and verifiable credentials (VCs) on a permissioned blockchain; (iii) a Private Resource-Ranking Service that employs functional encryption to match encrypted resource offers against application QoS goals without disclosing sensitive descriptors; and (iv) a Decentralised Knowledge Base (KB) that stores and disseminates resource, monitoring and orchestration metadata in a decentralized manner.

The deliverable thereby fulfils its twofold objective: (1) to define trust and trust attributes suited to swarm-based orchestration, and (2) to provide an implementable, open architecture for turning that definition into real-time, on-chain evidence—all while safeguarding provider confidentiality and end-user privacy.

# 1. Introduction

Trust is a foundational requirement for distributed and decentralised systems in which independent and often competing entities collaborate to achieve shared objectives. In the absence of a central governing authority, each participant must be able to evaluate the intentions, capabilities, and historical behaviour of every other participant in order to make defensible decisions about data sharing, resource allocation, and long-term cooperation. Consequently, rigorous trust mechanisms underpin essential system qualities such as security, reliability, and user satisfaction. The proliferation of swarm-based cloud platforms, large-scale IoT deployments, and blockchain ecosystems has further complicated this landscape by introducing highly dynamic actors, intermittent connectivity, and adversarial threat models that push traditional, static trust frameworks beyond their limits.

Swarmchestrate addresses these challenges through a decentralised, agent-oriented orchestration framework that places micro-services exactly where they are most effective along the Cloud-to-Edge continuum. Its vision hinges on the ability of autonomous "resource-capacity providers" and "application swarms" to negotiate, deploy, and adapt without exposing confidential business data or over-centralising control. Achieving this vision demands a multidimensional trust model that can (i) capture identity, behavioural, and contextual evidence; (ii) operate at the scale of thousands of heterogeneous nodes; and (iii) deliver near-real-time assessments robust against failure and manipulation.

To meet these requirements, Swarmchestrate introduces a Knowledge-Management Layer—a tightly integrated yet modular ensemble of four systems. The Trust-Management System aggregates heterogeneous metrics and transforms them into trust scores that are cryptographically certified. The Identity and Role Management System manages the decentralised identities of all involved entities in the platform. Moreover it binds the trust credentials to decentralised identifiers and stores them on a permissioned blockchain, thereby ensuring transparency, and seamless smart-contract interaction. Complementing these components, a Privacy-Preserving Mechanism (Private Scoring Service) employs functional-encryption techniques to rank encrypted resource offers against application quality-of-service goals—revealing only the final ordering and keeping sensitive commercial data hidden. All raw telemetry, resource descriptors, and derived artefacts are maintained in a Decentralised Knowledge Base (KB) that combines off-chain document stores with an on-chain contributions log to guarantee provenance, availability, and fault tolerance. Together, these subsystems transform low-level signals into verifiable, privacy-respecting knowledge that the Swarmchestrate orchestrator can exploit to make auditable deployment and adaptation decisions.

Deliverable D3.1 "Definition of Trust and Initial Design of the Trust- and Privacy-Management Systems" formalises the notion of trust for the Swarmchestrate context and elaborates the architectural blueprint of the four subsystems that constitute the Knowledge-Management Layer. By defining measurable trust attributes, mapping them to concrete data flows, and specifying the interfaces among subsystems, the document provides the conceptual and technical foundation upon which subsequent prototypes and integrations will be built.

The remainder of this report is organised as follows: Section 2 reviews the project context, stakeholder roles, and high-level requirements. Section 3 introduces the multidimensional trust taxonomy and explains the corresponding attributes. Sections 4 through 7 detail the designs of the Trust-Management System, the Identity- and Role-Management System, the Privacy-Preserving Mechanism, and the Decentralised Knowledge Base, respectively. Section 8 synthesises these designs into the overarching Knowledge-Management Layer, highlighting internal and external interfaces, while Section 9 concludes with a summary of contributions and an outline of future work.

# 2. Swarmchestrate background information

Managing and analysing vast data in the Cloud-to-Edge continuum poses significant challenges. Centralized cloud processing is increasingly impractical due to data transfer costs, latency, performance degradation, and security concerns. To address this, edge and fog computing paradigms enable localized data processing but demand novel approaches to manage hyper-distributed systems.

Swarmchestrate introduces a decentralized application orchestrator based on self-organizing swarms [1]. Application microservices are dynamically managed in an orchestration space by decentralized agents using artificial intelligence for resource management and swarm self-organization. Trust and security is achieved through the use of blockchain-based technologies Self-Sovereign Identities (SSI) and Decentralized Identifiers (DID), while advanced cryptography ensures system-wide security.

The Swarmchestrate concept will be prototyped on four real life demonstrators from the areas of flood prevention, parking space management, urban noise classification and a digital twin of natural habitat.

Swarmchestrate's objectives are as follows:

- Develop an application-level decentralised orchestration framework (incorporating compute, data and code), utilising Swarm-based distributed intelligence for highly dynamic and distributed Cloud-to-Edge computing infrastructures.
- Dynamically create and manage a set of interconnected Swarms by matching application requirements with resources across the distributed Cloud-to-Edge infrastructure.
- Develop matchmaking algorithms using decentralised AI methods that will dynamically pair application requirements with resource characteristics, in order to optimise energy efficiency, resilience and effectiveness of the system.
- Develop a trusted, reliable, secure and transparent knowledge management infrastructure.
- Develop a simulation environment based on the novel decentralised orchestration concept of the project that will be utilised to measure and finetune the effectiveness of the implemented solutions and algorithms. Additionally, further extend this simulator to be used to create a digital twin of the orchestration solution, supporting the live system by recommending predictive modifications of the system setup and its parameters.
- Implement four real-life application demonstrators utilising Swarmchestrate services in realistic scenarios where large amounts of data, collected at the network edges, need to be efficiently processed.

Swarmchestrate recognizes six different types of entities: resource, resource provider, capacity, capacity provider, users and application users.

- Resource is any computational resource ranging from a dynamically created virtual machine in the cloud to a physical node at any layer of the compute continuum, to an intelligent sensor with processing capabilities.

- **Resource Provider** is the provider of the resources where the resources run. For instance, in the case of cloud resources, cloud provider such as Amazon is resource provider. In the case of edge devices, it is the owner of the devices.
- **Capacity** refers to the logical grouping of resources offered (made available) by an entity within the Swarmchestrate universe. For example, Acme Corporation owns four edge devices in London, has access to 100 CPUs and 512 GB memory on AWS, and has access to 48 CPUs and 1024 GB memory on Azure. All these different sets of resources are from different resource providers, however, within Swarmchestrate, all these may be made available as one or more logical groups (capacities) by Acme.
- **Capacity Provider** refers to the provider of the capacity. E.g. Acme Corporation in the above-mentioned example.
- **Users** are the application operators.
- **Application Users** are the users of the applications hosted by the Swarmchestrate, however, this type of user and their requirements are not within the scope of the project.

# 3. Definition of trust in Swarmchestrate

## 3.1 General definition of trust

Trust is a foundational concept that permeates social, economic, and technological interactions. It embodies the willingness of one entity to rely on another's actions in uncertain situations, where outcomes are dependent on the trusted party's behavior. Despite its ubiquity, trust is notoriously difficult to define comprehensively due to its multifaceted and context-dependent nature.

The Oxford English Dictionary defines trust as "*the belief that somebody/something is good, sincere, honest, etc. and will not try to harm or trick you*", or "*the belief that something is true or correct or that you can rely on it*".

In the following are several definitions of trust, showcasing its multifaceted nature throughout the years across different disciplines and perspectives:

- Rotter's Definition (1971) [2]: Trust is "*a generalized expectancy held by an individual or group that the word, promise, verbal, or written statement of another individual or group can be relied on.*" This definition emphasizes trust as a relatively stable personality characteristic;
- Gambetta Definition (1988) [3]: Trust *(or, symmetrically, distrust) is a particular level of the subjective probability with which an agent assesses that another agent or group of agents will perform a particular action, both before he can monitor such action (or independently of his capacity ever to be able to monitor it) and in a context in which it affects his own action.* Trust and reputation *are considered 'subjective probabilities by which an individual, A, expects that another individual, B, performs a given action on which its welfare depends'.* Trust and reputation are not the result of a mental state of the agent in a cognitive sense but the result of a more pragmatic game with utility functions, and numerical aggregation of past interactions.
- Coleman's Rational-Choice Definition (1990) [4]: Trust is a calculated decision where "*the elements confronting the potential trustor are nothing more or less than the*

*considerations a rational actor applies in deciding whether to place a bet.*" It ties trust to risk and rational action;

- Barney and Hansen's Definition (1994) [5]: Trust is "*the mutual confidence that no party to an exchange will exploit another's vulnerabilities.*" This definition situates trust within the dynamics of exchange relationships, distinguishing between weak-form, semi-strong-form, and strong-form trust;
- Holton's Definition (1994) [6]: Trust involves "*reliance on someone to do something, accompanied by readiness to feel betrayal should that reliance be disappointed and gratitude should it be upheld.*" This account emphasizes the affective and normative dimensions of trust;
- Marsh's Computational Trust Definition (1994) [7]: Trust is categorized into three types: *basic trust (general disposition), general trust (non-situation-specific trust between agents), and situational trust (context-specific trust derived from utility and importance in a given interaction)*.
- Mayer, Davis, and Schoorman's Definition (1995) [8]: Trust is "*the willingness of a party to be vulnerable to the actions of another party based on the expectation that the other party will perform a particular action important to the trustor, irrespective of the ability to monitor or control the party.*" This definition highlights vulnerability as a key component of trust;
- Lewicki and Bunker's Definition (1995) [9]: Trust is "*a state involving confident positive expectations about another's motives regarding oneself in situations of risk.*" This definition categorizes trust based on its sources, such as calculus-based, knowledge-based, and identification-based trust;
- Bhattacharya, Devinney, and Pillutla's Definition (1998) [10]: Trust is "*an expectancy of positive (or nonnegative) outcomes that one can receive based on the expected action of another party in an interaction characterized by uncertainty.*" This statistical and mathematical perspective aligns trust with expectancy under uncertain conditions;
- Grandison and Sloman's Definition (2000) [11]: Trust in Internet applications is defined as "*the degree to which an entity is willing to depend on another entity in a given situation with a feeling of relative security, even though negative consequences are possible.*" This definition is tailored for computational and digital environments;
- McKnight and Chervany's Definition (2000) [12]: *Trust is a broad, generic concept vital to mitigating uncertainty and fostering interdependence in relationships. It is characterized by constructs such as competence, benevolence, integrity, and predictability*;
- Esfandiari and Chandrasekharan Definition (2001) [13]: "*Trust and reputation are made up of underlying beliefs and are a function of the degree of these beliefs*". In the cognitive approach, the mental states that lead to trust another agent or assign a reputation, as well as the mental consequences of the decision and the act of relying on another agent, are an essential part of the model.
- Simpson's Genealogical Definition (2012) [14]: Trust arises from "*basic features of what it is for humans to live socially, in which we rely on others to act cooperatively.*" Simpson describes trust as a concept with nuances shaped by hope, threat, and the need for cooperation;

These definitions collectively illustrate trust's diverse interpretations, emphasizing its adaptability to various contexts such as interpersonal relationships, organizational structures, and computational systems.

Since it is evident that there is lack of global trust, i.e. there are no stakeholders everybody trusts, few natural characteristics follow:

- Trust is not symmetric: If "Alice trusts Bob," it does not necessarily mean that "Bob trusts Alice."
- Trust is not distributive: If "Alice trusts (Bob and Carol)," it does not imply that "(Alice trusts Bob) and (Alice trusts Carol)."
- Trust is not associative: The trust operator does not map from entities to entities. Therefore, "(Alice trusts Bob) trusts Carol" is not a valid trust expression. However, "Alice trusts (Bob trusts Carol)" is a possible scenario.
- Trust is not inherently transitive: If "Alice trusts Bob" and "Bob trusts Carol," it does not automatically follow that "Alice trusts Carol."

In the Cloud-to-Edge computing continuum, trust management must move beyond monolithic, provider-centric models and instead cope with a continuum where no single entity is universally trusted. The novelty lies in orchestrating fine-grained, dynamic trust decisions among highly heterogeneous, mobile, and intermittently connected entities. Trust must therefore be treated as a situational, continuously updated metric, blending computational assessments with subjective probabilities and risk-aware rational choice. Because trust is neither transitive nor distributive, the continuum demands mechanisms that evaluate each interaction path on-the-fly, incorporate local context (latency, resource constraints, data sensitivity and etc.), and reconcile multiple trust dimensions (competence, integrity, identity and etc.) across administrative domains. This adaptive, multi-party, context-aware trust fabric represents the critical innovation enabling secure, dependable collaboration from centralized clouds down to all the participants in the continuum.

## 3.2 Trust in the Cloud-to-Edge continuum

In technological contexts, trust is a measure of the assurance stakeholders have in a system's ability to fulfill its intended purpose. It is particularly critical in environments where systems are autonomous and decentralized, such as swarm-based cloud ecosystems and IoT networks. Trust in technology extends beyond human-to-system interactions to include system-to-system and peer-to-peer relationships, where entities must evaluate the trustworthiness of their counterparts dynamically.

In other words, the concept of trust in the computing continuum revolves around the confidence users and stakeholders place in the interconnected systems that connect physical "things" (e.g., IoT devices) to cloud-based infrastructures. This trust is essential as these systems power various applications in smart cities, healthcare, industrial automation, and many more. It depends on multiple critical attributes, including security, privacy, reliability, transparency, data integrity and similar. These attributes will be further explained in the sections below.

Considering the definitions above, *trust in the context of Cloud-to-Edge computing can be summarized as the subjective probability or expectancy that one agent or system has regarding another agent or system's ability to perform a particular action reliably and securely, especially*

*in situations where monitoring is not possible and the outcome significantly impacts the trusting agent. This trust is influenced by a combination of stable personality characteristics, rational calculations of risk, mutual confidence in exchange relationships, affective and normative dimensions, and the context-specific expectations of positive outcomes under uncertain conditions.*

This definition can be considered as the baseline definition of trust in Swarmchestrate, its components will be described in details in the following sections.

## 3.3 Trust in Swarmchestrate

The trust in Swarmchestrate can be depicted through a taxonomy that lays out a five-pillar taxonomy that follows the life-cycle of trust from the moment it is invoked to the mechanisms that actively strengthen (see Figure 1). The lifecycle of trust begins with its timing—when and how trust is evaluated—spanning the immediate function trust needed for transient decisions and the long-term experience trust built through repeated engagements. This lifecycle continues with the trust acquisition, which involves mechanisms for assessing trustworthiness through direct interactions and indirect sources, such as reputation-based and contextual trust. However, effectively utilizing acquired trust across a system requires a robust framework for trust representation, which translates trust assessments into actionable formats. Trust representation ensures that trust values, whether derived from subjective perceptions or objective metrics, are consistently communicated and understood, forming the foundation for decision-making and collaboration.

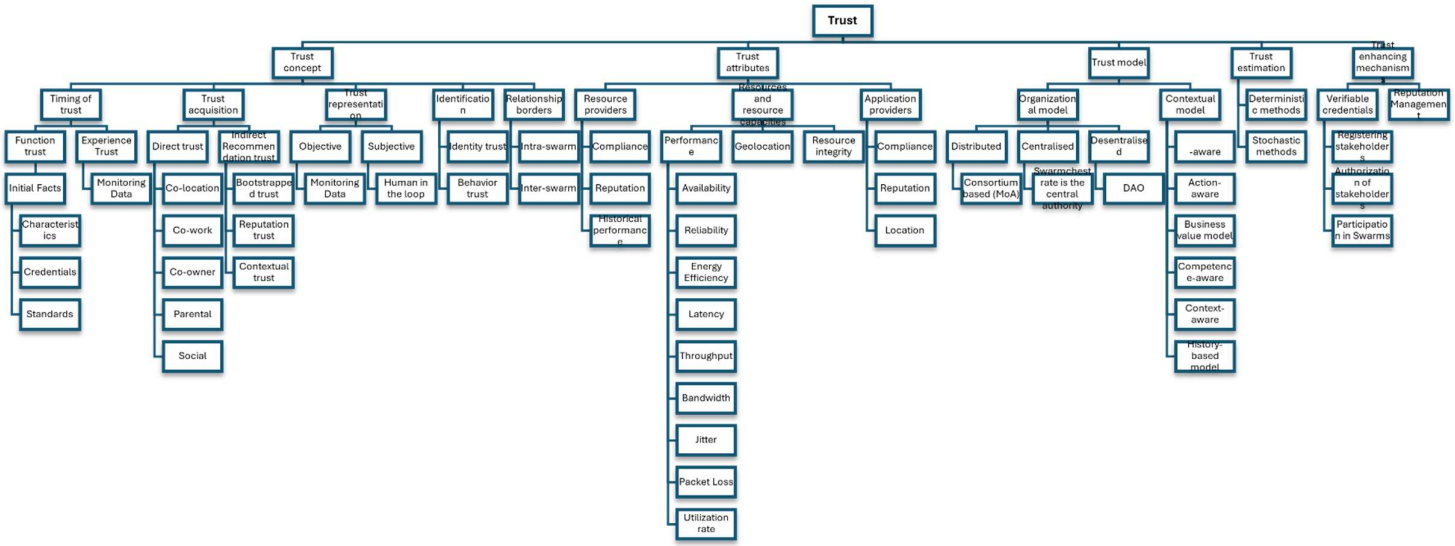



*Figure 1 Taxonomy of Trust* [1]

[1] Full size figure is available in the Annex of this document.

### 3.3.1 Timing of trust

The timing of trust establishes confidence in system interactions across all involved stakeholders such as IoT devices, edge nodes, and cloud systems. Timing of trust refers to the temporal aspect of trust evaluation, encompassing how and when trust is established, used, and maintained over time. It is primarily concerned with the lifecycle of trust in interactions. This concept is particularly relevant in environments such as Swarmchestrate, where distributed resources dynamically collaborate. The timing of trust integrates two key components: Function Trust and Experience Trust, each grounded in distinct foundations—initial facts and monitoring data, respectively.

Function trust is built on immediate, observable characteristics and credentials, making it the starting point for trust evaluation in a swarm-computing context. This form of trust is crucial for enabling rapid decision-making during initial interactions between entities.

Function trust is derived from initial facts such as: characteristics (e.g. resource characteristics, energy requirements, computational capabilities and etc.), credentials (e.g. digital identities, authentication tokens) or standards' compliancy (e.g W3C, IEEE). Although function trust covers wide spectrum of facts that can be used as baseline for establishing trust, it still has limitations because it evaluates entities based on predefined properties rather than adaptive insights.

Experience trust evolves over time through the analysis of monitoring data, offering a dynamic and adaptive layer of trust. It is a long-term process, relying on patterns and behaviors observed over time. The experience trust enables systems to refine trust relationships based on observed performance and behaviors. The trust is built upon behavioral data (e.g. swarm interaction logs, communication consistency, or error frequencies) and monitoring metrics (e.g. network performance and computational performance). Although this type of trust is of dynamic nature, its limitation is that the monitoring system will have to handle large volumes of data in real time. Moreover, trust may decay over time if behaviors are not consistently monitored and analyzed.

### 3.3.2 Trust acquisition

Trust acquisition focuses on the methods and mechanisms used to establish trust. It involves collecting and processing information to assess the trustworthiness of entities. Trust acquisition encompasses two main categories: direct trust and indirect trust, with the latter further divided into reputation-based trust and contextual trust. Together, these components create a multi-layered framework to assess and manage trust dynamically.

Direct trust is established through direct, observable interactions between entities in the system. It is a vital initial step, as it enables the system to gauge trustworthiness based on real-time engagements rather than relying solely on secondary information. Authentication mechanisms such as digital identifiers, digital certificates, cryptographic keys, and password credentials ensure that only legitimate entities can participate in Swarmchestrate. Performance data, such as computational performance and network performance can be used to evaluate trustworthiness during active interactions. Observing stakeholder's behavior over time also strengthens direct trust by identifying consistent patterns of reliability, thereby refining trust decisions.

Establishing such relations among the stakeholders can be based on different criteria such as resource specifications, activity patterns, applications hosted and etc. Based on this the following relationships can be established [15]:

- **Co-location relationship** is established between stakeholders that are in the same location, such as within same company, country or wider geographical territory that abides the same laws and regulations (e.g. GDPR).
- **Co-work relationship** is established between stakeholders that collaborate to offer a common service (e.g. capacities within a swarm hosting the same application or services).
- **Parental relationship** is established between IoT devices that were produced in the same manufacturing batch (i.e. manufacturer, period and model).
- **Social relationship** is the trust established by the resources when they come in contact with each other due to the resource providers being part of the same ecosystem.
- **Co-owner relationship** is established between different resource capacities that belong to the same provider.

However, direct trust has also some limitations. For instance, it can only be established after an entity has interacted with others, making it insufficient for new or unknown devices. Moreover, malicious entities may initially exhibit trustworthy behavior to pass direct trust evaluations before engaging in harmful activities. To resolve these limitations, the direct trust is usually complemented by the indirect trust.

**Indirect trust** is established through secondary sources rather than direct interaction. This category of trust is applicable in the context of Swarmchestrate, where stakeholders often collaborate without prior interaction. Indirect trust can be divided into bootstrapped trust [16], reputation-based trust and contextual trust.

- **Bootstrapped trust** refers to the initial assignment of a nominal trust value to new stakeholders entering a system. In dynamic and decentralized systems such as Swarmchestrate, bootstrapped trust ensures that new stakeholders are not unfairly disadvantaged due to their lack of historical interactions or reputation. This approach fosters inclusivity and prevents the exclusion of potentially valuable newcomers.
- **Reputation-based trust** assesses the stakeholder's reliability based on aggregated historical data and feedback from other participants in the system. It can be particularly effective in environments like Swarmchestrate, where stakeholders constantly interact and collaborate. Reputation acts as a cumulative record of a stakeholder's performance, emphasizing consistency and reliability over time. This mechanism relies heavily on reputation scores, which quantify trustworthiness based on metrics such as Quality-of-Service, Quality-of-experience, success rates, compliance with protocols, responsiveness and etc. Despite its strengths, reputation-based trust faces a couple of challenges. Firstly, malicious entities might collude to provide false feedback, artificially inflating or deflating reputation scores. Secondly, older feedback may no longer accurately reflect stakeholder's current behavior, necessitating mechanisms to prioritize recent interactions. In such cases, time-weighted reputation models can give more significance to recent feedback while gradually diminishing the impact of outdated evaluations.
- **Contextual trust** complements reputation-based trust by evaluating an entity's trustworthiness based on real-time situational factors rather than historical data. It is

dynamic and adapts to the specific conditions of each interaction, making it particularly valuable in environments where entities lack sufficient historical records. This type of trust relies on analysing contextual parameters, such as geographic location, monitoring data (e.g. computing resource utilisation, network status, performance, energy consumption, and etc). Contextual trust is especially useful for addressing transient or time-critical challenges. For example, in a time-critical response systems, resources with optimal contextual trust may be prioritized for emergency tasks, regardless of their historical reputation. This adaptability ensures that trust evaluations align with immediate operational needs. However, contextual trust has also limitations in terms of its transient nature and may vary significantly based on external conditions, thus balancing contextual trust with reputation-based trust is therefore critical to achieving a robust and scalable trust framework.

### 3.3.3 Trust representation

Trust representation is the process of modelling, quantifying, and expressing trust in a way that enables consistent trust application. Depending on the context, the trust representation can be subjective or objective. These two perspectives represent different ways of conceptualizing and evaluating trust in systems, and they both play important roles in various environments

Objective representation of trust refers to models and frameworks where trust is evaluated based on measurable, verifiable data or metrics that are not influenced by personal opinions or biases. In these models, trust is typically quantified using explicit, predefined criteria, and the evaluation process follows clear, logical rules that are consistent and repeatable. For instance, the use of verifiable credentials in Swarmchestrate will be able provide an objective, verifiable basis for trust of the various stakeholders (e.g. resource providers, resource capacities), as the validity of the certificates can be securely and transparently verified. For that purpose, Swarmchestrate's monitoring system and decentralized knowledge base will be responsible for delivering the necessary data for objective trust estimation.

Subjective representation of trust is based on personal judgments, experiences, or beliefs about the trustworthiness of another entity. This type of trust often involves opinions, interpretations, or feelings about a particular agent, and it may be influenced by past experiences, emotional factors, or individual perspectives. Trust is not always quantifiable in this case, and different agents or individuals may have different levels of trust based on subjective factors. The direct trust and the reputation trust are examples of subjective representation because these models are in nature based on individual experiences.

In many cases, trust models combine both subjective and objective elements to create a more comprehensive and adaptable system. A hybrid approach can balance the precision and consistency of objective trust with the flexibility and nuance of subjective trust. This is also the case in Swarmchestrate where on one side the monitoring system and decentralized knowledge base will participate in the objective trust modelling, whereas on the other side the human in the loop (e.g. a central trusted authority in the platform) can deliver a subjective trust model as part of the direct trust.

### 3.3.4 Identification for trust

Identity Trust refers to the trust that is placed in an individual, system, or entity based on its identity credentials and the ability to prove its legitimacy. This type of trust is foundational in

establishing whether an entity is who they claim to be. Identity trust is established through authentication mechanisms. The core of identity trust lies in the assurance that a stakeholder or system is the rightful entity it claims to be, without the risk of impersonation or fraud. For example, when resource providers in Swarmchestrate want to register a resource capacity in the system, they use decentralized identifier and verifiable credential and trusts that the system will correctly identify them based on the credentials they provide. The level of trust in identity depends on the robustness of the authentication mechanisms and the reputation of the identity verification process.

Behavior Trust, on the other hand, is the trust that is built over time through consistent, predictable actions of a stakeholder. It is not enough for a stakeholder to prove their identity once; their actions, decisions, and interactions are continuously observed to assess the trustworthiness of their behavior. In essence, behavior trust is about observing and analyzing how someone or something behaves in practice, and this is often referred to as reputation-based trust. For example, Swarmchestrate might trust a resource capacity based on its history of positive interactions or high reputation. Behavior trust is dynamic and evolves over time, which means it is more fluid compared to identity trust. Stakeholders can build or lose trust based on how their actions align with established norms, standards, and the expectations of others. In the digital age, behavior trust is often linked to algorithms that track and analyze patterns of activity, and it can be influenced by factors such as past experiences, reputation scores, or ratings.

### 3.3.5 Relationship borders

In any distributed or decentralized system, managing trust between different groups of entities is essential to maintaining security, efficiency, and collaboration. Relationship borders define the boundaries within which these entities interact, both within a single group or system (intra-swarm) and between different groups or systems (inter-swarm). These boundaries ensure that trust is managed appropriately, maintaining the integrity of the overall system while allowing for cooperation across various components.

Inter-Swarm Relationship Borders refer to the boundaries and trust mechanisms that govern interactions between different swarms within Swarmchestrate. These swarms can be seen as distinct clusters of computing nodes that may belong to different organizations, but still need to communicate, cooperate, or share information. To assure trust, each swarm must authenticate the identities of the resource capacities in the other collaborating swarm (identity trust) and ensure that the behavior of these entities aligns with the established rules, protocols, and expectations (behavior trust).

Intra-Swarm Relationship Borders, on the other hand, focus on trust mechanisms within a single swarm. This pertains to how different resource capacities within the same swarm interact and rely on each other to ensure the smooth functioning of the collective. Even though the components of an intra-swarm system may be part of the same overall group, they must still maintain clear boundaries regarding authority, access, and behavior.

For instance, within a single swarm, agents may need to interact to complete tasks, share information, or ensure security within the swarm itself. These interactions are based on trust in both identity (ensuring that only authorized agents can perform specific functions) and behavior (ensuring that actions taken by agents align with the swarm's expected norms). Intra-swarm trust is typically defined through mechanisms like access control, role-based

permissions, and continuous monitoring of actions to ensure all entities within the swarm behave as expected.

### 3.3.6 Trust model

A trust model is a framework or methodology that defines how trust is established, managed, and evaluated among the stakeholders within a system. Trust models evaluate the trust attributes to enable collaboration while mitigating risks.

Trust models can be classified into two broad types based on their organizational approach and the specific dimensions of trust they evaluate. The first type includes centralized, decentralized, and distributed models, while the second type evaluates trust from perspectives such as identity, actions, business value, and context.

Centralized trust models rely on a single authority or entity to establish and manage trust across the system. This authority acts as the arbiter for identity verification, trust evaluation, and decision-making. Centralized models are straightforward to implement and offer a single source of truth, ensuring consistent trust evaluations across the system. This model is most fitting for Swarmchestrate because the system will be considered as a central point of truth.

In decentralized models, trust management is distributed among multiple different stakeholders, typically through peer-to-peer (P2P) interactions. Decisions are made collaboratively without relying on a central authority. They use technologies such as blockchain, DAOs, DID, VCs to ensure transparency, privacy and reliability.

Distributed trust models eliminate centralized control entirely, distributing all trust-related processes across the system. Each entity evaluates trust based on local observations, shared data, and collective reputation.

Identity-aware models establish trust based on the verified identity of stakeholders. They ensure that only authenticated and authorized devices or users can participate in the system. This is particularly important in systems such as Swarmchestrate where rogue devices could compromise system integrity. Understanding identity does not always mean knowing someone's real name or globally unique details. In some cases, it is enough for their identity to be unique within a specific context and consistent over time, so they can be recognized as the same person in that setting. For instance, within Swarmchestrate, DIDs are used to identify stakeholders without revealing their identity details.

Action-aware trust models evaluate stakeholders based on their behaviors and actions over time. Stakeholders are trusted if their actions align with predefined expectations or acceptable norms. In action-aware models, the set of actions can be closed or open. A closed set is a set of pre-defined actions the model supports and no others can be defined by organizations using the model. An open set of actions means that the model offers a way of defining at least some of the actions or does not restrict them to a particular set. For example, in Swarmchestrate, all stakeholders will be assigned activities based on their role in the system and the stakeholders that cooperate and perform their assigned tasks successfully will build a higher trust score.

Business value models focus on the business impact of trust decisions, assigning higher trust to stakeholders that contribute more significantly to the organization's goals or financial outcomes. In this model, concepts like risk, benefit, and value are used to evaluate how an action could either increase or decrease trust. While risk is the most modelled element,

understanding an action's full impact requires balancing risk with potential benefits. Together, these factors are unified under the concept of business value, representing the overall positive or negative impact of an action.

Competence-aware models assess a stakeholder's capability to perform its assigned tasks. In Swarmchestrate, resource capacities fulfilling the predefined requirements that have a history of accurate and timely task completion can be trusted more than less competent stakeholders. On the other hand, capability-aware models go beyond competence by evaluating the potential of a stakeholder to handle specific tasks or workloads.

Context-aware trust models consider the situational factors influencing trust. In this model, trust depends on the evaluating system internal or external status at that particular point of time, i.e. context. These models adapt trust evaluations dynamically based on real-time contextual changes.

History-based trust models rely on past interactions and behaviors to evaluate trust. Entities with a positive history of collaboration, reliability, or integrity are assigned higher trust scores. This can be modeled using vocabulary like *experience* or *evidence* or *local reputation*. For example, a device consistently reporting accurate sensor data would be trusted more over time.

The trust model implemented in Swarmchestrate embodies a hybrid approach to trust management, aligning with the project's objectives to establish a central authority for access and role management within the system, whilst exploiting the transparency, traceability and security benefits of its decentralized components (i.e. the decentralized knowledge base and the blockchain). The Swarmchestrate platform ensures secure and efficient orchestration of resources by maintaining full control over identity and role management. This centralized authority simplifies decision-making and fosters consistency in trust evaluation across the dynamic, multi-stakeholder environment. Trust within the system is dynamically established based on three key pillars: real-time monitoring data, identity and role management, and prior usage history. Real-time monitoring data, provided by the system's integrated monitoring components during orchestration and runtime, ensures that trust evaluations are reflective of the system's current state. Identity and role management ensures that only authorized and verified stakeholders participate, establishing a baseline for secure collaboration. Additionally, prior usage data enables the implementation of action-aware, competence-aware, context-aware, and history-aware trust models. These contextual trust dimensions enhance decision-making by incorporating the behavioral, situational, and historical performance of stakeholders into trust evaluations.

Figure 2 presents a high-level representation of a mechanism designed to implement the previously described trust model. At its core, the blockchain layer ensures transparent and secure interactions within the system, facilitating the management of access and the assessment of trust attributes for various stakeholders. Complementing this, the decentralized knowledge base, stores crucial data related to trust attributes. This data is integral for calculating the trust levels of the involved participants in the system, enabling a robust and reliable trust assessment process throughout the system.

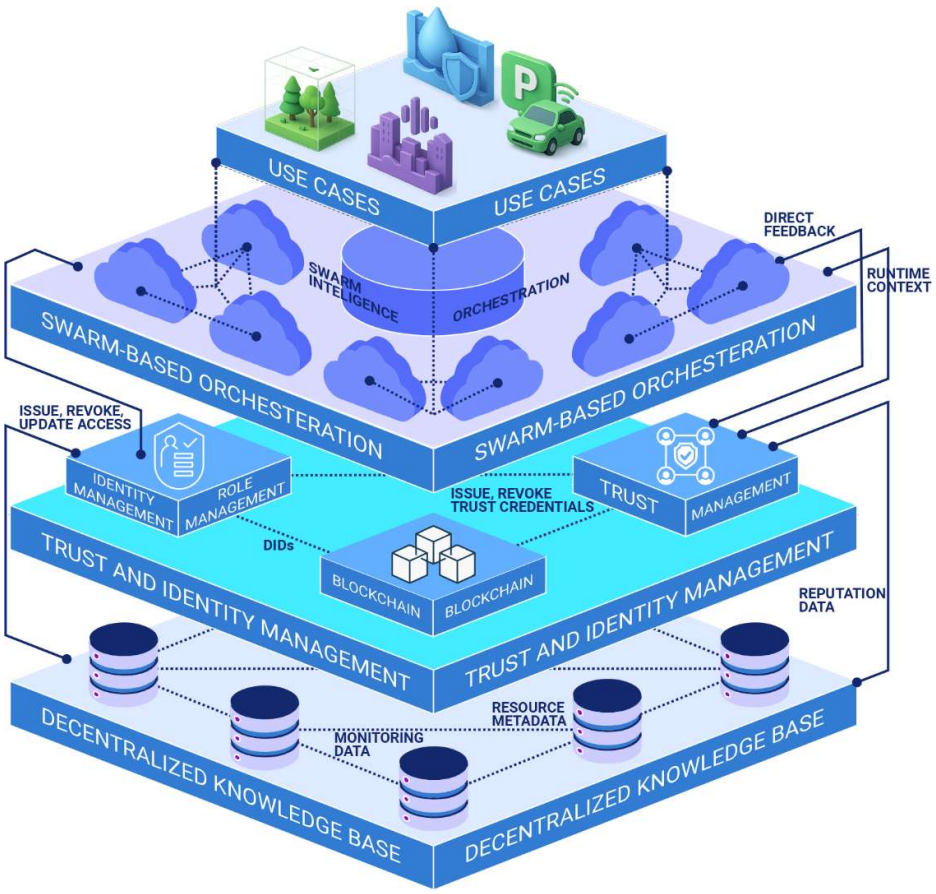



*Figure 2 High-level trust model representation*

### 3.3.7 Trust enhancing mechanisms

Swarmchestrate implements robust trust-enhancing mechanisms through two primary components: the Trust Management System and the Identity and Role Management System. These components work together to ensure a secure, transparent, and trustworthy ecosystem for all entities involved. The trust management system will calculate the trustworthiness of entities within Swarmchestrate, such as application providers, resource providers, and resource capacities. It leverages blockchain technology, DIDs and VCs to provide a transparent and immutable record of trust-related activities, ensuring secure interactions between entities. By enforcing predefined trust criteria, the system reduces risks and promotes confidence in the network.

The identity and role management system focuses on the secure and efficient handling of identities and roles within Swarmchestrate. It establishes cryptographically secure identities using DIDs, VCs, and SSI. This ensures that all entities joining Swarmchestrate are empowered with reliable and tamper-proof digital identities. This system also supports role-specific operations, defining and enforcing permissions tailored to each entity's function, such as application or resource providers. By implementing privacy-preserving methods, it enables traceable yet secure operations that enhance accountability without compromising user

privacy. Additionally, it simplifies access management by clearly defining roles and automating enforcement mechanisms.

### 3.3.8 Trust attributes

Trust attributes are quantitative and qualitative metrics, which ensure that all entities can rely on the platform's functionality, security, and performance, fostering collaboration and mitigating risks. The current list, outlined below, represents an initial version and will be further refined and fully defined in the upcoming phase of the project. The attributes used for trust calculation will be sourced from the decentralized knowledge base.

Resource providers and capacity providers:

- Compliance
- Historical behaviour
- Reputation

Resources and resource capacities:

- Placement
  - Location
  - Provider
- Performance
  - Availability
  - Reliability
  - Energy efficiency
  - Latency
  - Throughput
  - Jitter
  - Packet Loss
  - Bandwidth
- Reputation
  - User (i.e. application operator) satisfaction
  - Rating from capacities within the swarm
  - Platform (Swarmchestrate) rating

Users:

- Compliance
- Location
- Reputation (i.e. prior performance)

# 4. Initial design and implementation of a trust management system

## 4.1 System design

The Trust Management System (see Figure 3) is a self-contained service that is composed of the Trust Metric Aggregator, Trust Calculator and Trust Certifier. Whenever trust has to be calculated for a capacity or a provider, the Trust Metric Aggregator will first gather data coming from the other Swarmchestrate subsystems (e.g., Identity Management and Knowledge Management). It normalizes these heterogeneous data into a common JSON bundle as aggregated trust attributes, ensuring they share a consistent schema and timestamping so downstream components can treat the data uniformly.

That data flows into the Trust Calculator, which supports two independent algorithms for trust computation: a deterministic algorithm and a stochastic algorithm. The platform can invoke either algorithm separately to calculate trust. The resulting trust value is output in JSON format, persisted in the knowledge base, and subsequently used as input for issuing a verifiable credential for trust.

Finally, the Trust Certifier consumes that result and transforms it into a cryptographically signed, verifiable credential. By decoupling certification from calculation, the design keeps cryptographic concerns isolated and allows the credential to be verified later—by any relying party—without re-running the algorithms or exposing raw evidence. In short, the figure highlights a clear separation of concerns: aggregation of evidence, pluggable computation, and immutable attestation, all packaged behind a single Trust Management System interface that the rest of Swarmchestrate can utilize to achieve high levels of trust.

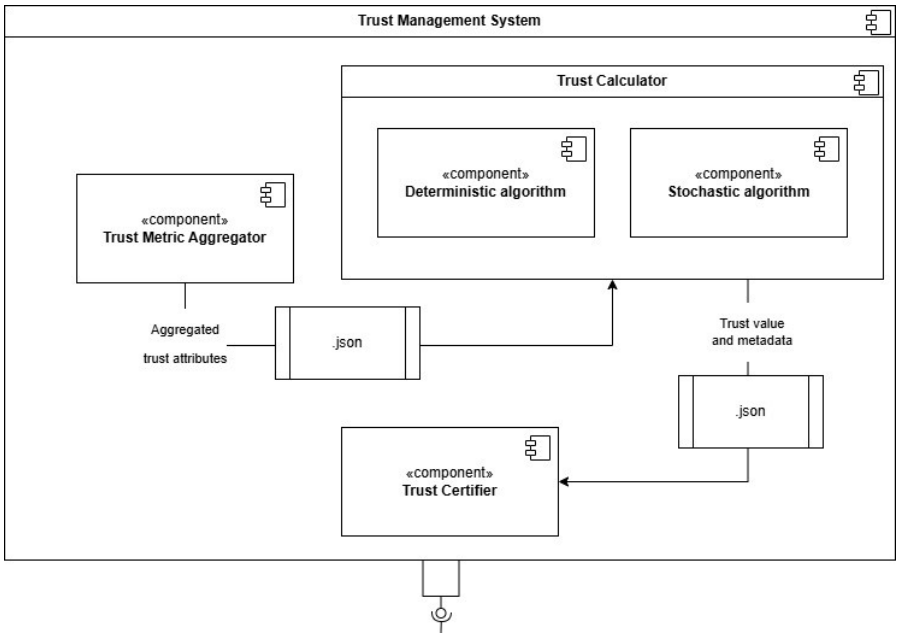



*Figure 3 Initial design of the trust management system*

## 4.2 Trust algorithms

The trust estimation can be done with multiple methods, which can be classified in two general groups: deterministic and stochastic.

### 4.2.1 Implementing the trust attributes

Before introducing the two trust-evaluation algorithms, it is essential to specify the trust attributes that are used as input. The trust attributes are classified into two distinct categories: conditional attributes and calculable attributes.

Conditional attributes act as Boolean gatekeepers that enforce mandatory SLOs. If a capacity or a provider violates any conditional attribute, it is immediately designated untrustworthy and removed from further processing—this exclusion is enforced uniformly by every algorithm. For example, geographic location can serve as a conditional attribute: entities operating from prohibited regions are rejected a priori.

Calculable attributes, by contrast, are quantitative inputs to the trust-scoring function applied after all conditional checks have passed. Their numerical values are mapped to trust contributions which, when aggregated, yield the final score. Network latency illustrates this class: lower latency typically elevates the score, whereas higher latency diminishes it. Should a strict latency ceiling be required, latency can be reclassified as conditional, disqualifying entities that exceed the threshold before the scoring phase begins.

This two-phase structure affords considerable flexibility. An attribute may first serve as a coarse exclusionary filter and later re-emerge as a continuous factor with fine-grained weight. Crucially, the trust-computation algorithms are executed only after the evaluation of conditional attributes is complete.

### 4.2.2 Normalizing Calculable Attributes

The Trust Calculator component produces a continuous trust score in the closed interval [0,1], where 1 denotes maximum confidence. Because the raw values of calculable attributes rarely fall inside this interval, each attribute must be normalised by referencing its expected minimum and maximum limits together with its intended influence on trust. Three canonical behaviours are supported.

- Positive correlation (behaviour = +1): When an attribute is positively correlated with trust, values at or below the specified minimum are mapped to 0 (indicating no trust contribution). Values at or above the maximum are mapped to 1, providing full trust. Intermediate values are passed through a monotonically increasing, Sigmoid-like transfer function that compresses the raw domain smoothly into [0,1], where higher values within the defined range contribute more positively to trust.
- Negative correlation (behaviour = −1): For attributes whose desirability decreases with magnitude (e.g., latency or error rate), the mapping is inverted. Values at or below the minimum translate directly to 1, whereas values at or above the maximum collapse to 0. Within the [min,max] span an inverted sigmoid provides a smooth decay, so that lower raw values yield higher trust and vice-versa.

- Optimal-range correlation (behaviour = 0). Some attributes exhibit maximal trust only when they lie near a preferred midpoint. Values that fall outside the prescribed range are immediately mapped to 0. Inside the range, trust is computed as a quadratic function of the distance from the midpoint: it peaks at the exact centre (value approaching 1) and decreases symmetrically towards both boundaries, reaching 0 at the limits. This quadratic profile penalises deviations while still allowing graceful degradation.

By selecting the appropriate behaviour and parameterising each attribute with domain-specific minima and maxima, the algorithm can capture a wide variety of real-world dynamics while preserving a uniform [0,1] output that can be aggregated reliably across heterogeneous metrics.

### 4.2.3 Deterministic algorithm

For each stakeholder $S$, the trust $T_S$ is calculated as a weighted sum of multiple trust dimensions. Each dimension evaluates a specific aspect of the stakeholder's behavior or characteristics. Given this the general formula can be defined as:

$$T_s = \sum_{i=1}^{n} w_i \cdot f_i(S)$$

Where:

- $T_S$: The trust calculation for particular entity (i.e. resource provider, application provider, or a capacity).
- $w_i$ : Weight assigned to the i-th trust attribute. The sum of all weights equals 1. The weights assigned to each trust attribute may vary based on the type of system or operational environment.
- $f_i(S)$: The value of the i-th trust attribute for stakeholder $S$.
- $n$: Total number of trust attributes considered.

#### Trust calculation for Resource providers and capacity providers

Given the explanation in the previous sections, the trust for resource providers is based on factors such as identity verification, service performance, compliance, and historical behavior. The considered attributes are:

- Identity Verification ($f_1$): The authenticity of the resource provider's identity and the management of its role in the system.
- Compliance ($f_2$): Adherence to service level agreements (SLAs), regulatory standards, and security practices.
- Reputation ($f_3$): Third-party feedback or reputation scores from other participants in the system.
- Direct trust ($f_4$)

The formal model would be:

$$T_{RP} = w_1 \cdot f_1(RP) + w_2 \cdot f_2(RP) + w_3 \cdot f_3(RP) + w_4 \cdot f_4(RP)$$

**Trust calculation for Resources or capacities**

The considered attributes for the resources and resource capacities are:

- **Identity Verification (f1)**: The credibility and trustworthiness of the application provider's identity.
- **Performance ($f_2$)**: Adherence to performance requirements (i.e. computing and network performance): Availability, Reliability, Energy efficiency, Latency, Throughput, Jitter, Packet Loss, Bandwidth and similar.
- **Contextual Fit ($f_3$)**: The alignment of the resource capacity with the application requirements, including context-aware evaluations.
- **Reputation ($f_4$)**: Feedback or reputation scores from other participants in the system.
- **Direct trust ($f_5$)**

The formal model in such case would be:

$$T_{RC}=w_1 \cdot f_1(RC) + w_2 \cdot f_2(RC) + w_3 \cdot f_3(RC) + w_4 \cdot f_4(RC) + w_5 \cdot f_5(RC)$$

### 4.2.1 Stochastic algorithm

This second algorithm employs a probabilistic framework, modelling trust using a Beta distribution, which is dynamically updated through an Exponentially Weighted Moving Average (EWMA) mechanism, sensitive to observation volatility. This approach allows the system to continuously learn and adjust trust scores, reflecting both past behavior and recent changes, while also quantifying uncertainty.

*Beta Distribution as a Trust Model*

The Beta distribution is an ideal choice for modeling trust scores, as it is defined over the interval [0, 1], perfectly aligning with our desired trust score range. It is parameterized by two positive shape parameters, α (alpha) and β (beta). In the context of trust, these can be intuitively interpreted as the accumulated count of positive outcomes (e.g., trustworthy behaviors) and negative outcomes (e.g., untrustworthy behaviors), respectively.

The shape of the distribution directly visualizes the certainty of the trust assessment; as more evidence is gathered (increasing α and β), the distribution becomes more peaked and narrow, signifying higher certainty in the trust value.

The shape of the Beta distribution (see Figure 4) directly reflects the degree of certainty about the trust score. As more evidence accumulates (i.e., as α and β increase), the distribution becomes narrower and more peaked, indicating higher certainty in the estimated trust value. Conversely, low values for α and β (e.g., initial values of 1.0) result in a flatter, more uncertain distribution.

Methodologically, the Beta distribution is particularly powerful here because it is the conjugate prior for the Bernoulli and Binomial distributions. This means that if our prior belief about trustworthiness is in the form of a Beta distribution, and we observe new evidence (which can be modelled as a success/failure trial), the resulting posterior belief is also a Beta distribution.

This property makes updating our trust belief computationally efficient and elegant, which is a core feature of Bayesian inference.

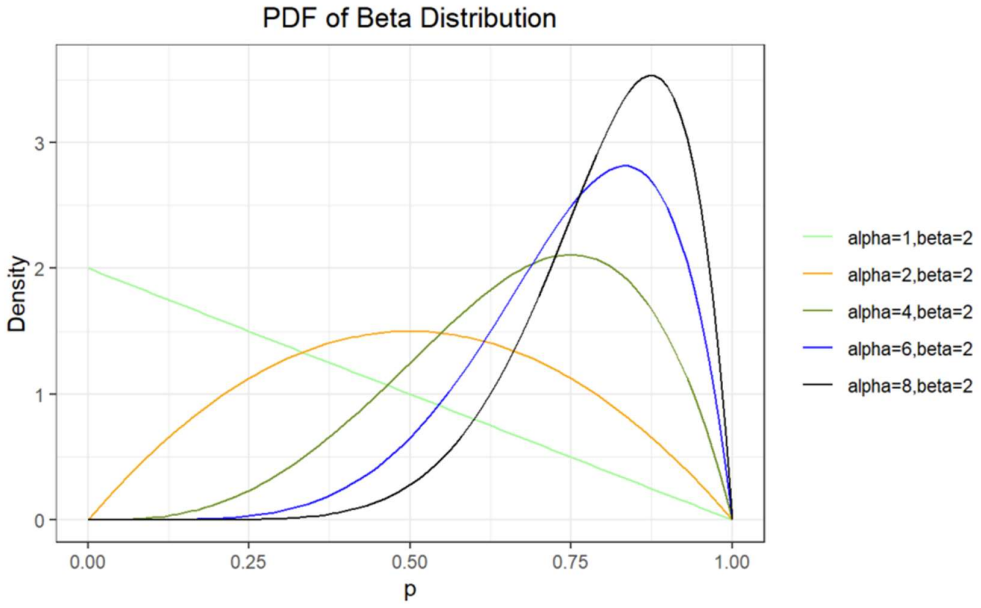



*Figure 4 Beta distribution*

### *Core Update Mechanism: Observation and Adaptive EWMA*

Each time a new observation (a normalized calculable attribute value, v ∈ [0, 1]) is made for a feature, the model updates its internal representation of trust. This update is managed by the 'observe' function, which processes the new data point and adjusts the α and β parameters of the Beta distribution.

The update rule combines the principles of an Exponentially Weighted Moving Average (EWMA) with an adaptive learning rate and an increasing effective sample size:

- **Effective Sample Size ($n_{eff}$)**: The parameter $n_{eff}$ acts as a proxy for the total amount of evidence accumulated. It starts at a low value (e.g., 2.0) to represent initial uncertainty and gradually increases with each observation. This growth is tempered by volatility, meaning high volatility slows down the accumulation of effective samples, reflecting less stable evidence. The update for $n_{eff}$ is given by:

$$n_{eff} \leftarrow n_{eff} + \frac{growth\_rate}{volatility + 1}$$

  Before updating with new observations, α and β are first scaled down by $1/n_{eff}$ to maintain their proportional influence relative to the growing effective sample size.

- **Adaptive Learning Rate (λ)**: A crucial component is the 'adaptive lambda' (or λ), which determines the weight given to the new observation versus the historical data. Unlike a fixed λ in traditional EWMA, this algorithm dynamically computes λ based on the observed volatility of the attribute's measurements.

$$\lambda = \max(0, \min(1, base\_lambda \cdot (1 + 2.5 \cdot volatility)))$$

Here, 'base lambda' is a predefined parameter. This adaptive nature ensures that if an attribute's behavior is highly volatile, new observations have a stronger and more immediate impact on the trust score, allowing for quicker adaptation to changes. Conversely, in stable environments, new observations have less sway, promoting stability in the trust estimate.

- **Updating α and β**: The core update for the Beta distribution parameters uses the adaptive λ and the new observation *v*:

$$\alpha \leftarrow ((1-\lambda) \cdot \alpha + \lambda \cdot v) \cdot n_{eff}$$
$$\beta \leftarrow ((1-\lambda) \cdot \beta + \lambda \cdot (1-v)) \cdot n_{eff}$$

Multiplying by $n_{eff}$ after the EWMA step ensures that as more experience is gained, the certainty of the distribution (represented by the magnitudes of α and β) increases.

*Volatility Estimation*

Volatility is estimated from a short history of recent observations. The algorithm maintains a limited window of the most recent measurements (e.g., 5 observations). The standard deviation of these recent measurements quantifies the volatility.

$$Volatility = Standard\ Deviation\ of\ Recent\ Observations$$

If fewer than two measurements are available, volatility is considered zero. This real time volatility assessment directly informs the adaptive learning rate, making the model responsive to changes in an entity's behavior.

*Trust Score Derivations and Uncertainty Quantification*

The model provides several metrics derived from the Beta distribution to fully characterize trust:

- **Trust Score (Expected Value)**: The primary trust score for a feature is the expected value (mean) of the Beta distribution, which represents the most probable trust value given the accumulated evidence:

$$Trust\ Score = E[Beta(\alpha, \beta)] = \frac{\alpha}{\alpha + \beta}$$

- **Uncertainty (Variance and Standard Deviation):** The model explicitly quantifies the uncertainty associated with the trust score using the variance and standard deviation of the Beta distribution. Higher variance indicates greater uncertainty in the estimated trust.

$$Variance = \frac{\alpha \cdot \beta}{(\alpha \cdot \beta)^2 (\alpha + \beta + 1)}$$
$$Standard\ Deviation = \sqrt{Variance}$$

**Adjusted Trust Score**: To provide a more conservative estimate that accounts for the model's confidence, an adjusted trust score is calculated by penalizing the primary trust

score by its uncertainty. This penalty is applied only after a sufficient number of effective samples (e.g., nef f ≥ 4) have been accumulated, to avoid penalizing initial, highly uncertain estimates too harshly.

$$\textit{Adjusted Trust Score} = \textit{Trust Score} \cdot (1 - \textit{uncertainty penalty} \cdot \textit{Standard Deviation})$$

The ”uncertainty penalty” is a configurable parameter that determines how severely uncertainty impacts the final score. The result is clamped between 0.0 and 1.0.

Confidence Interval: For a more comprehensive understanding, the model can also compute a confidence interval (e.g., 95% confidence interval) around the trust score. This interval provides a range within which the true trust value is expected to lie with a given probability, further illustrating the certainty of the estimate.

This probabilistic algorithm offers a dynamic, adaptive, and nuanced approach to trust evaluation, providing not just a score but also a measure of its reliability, making it suitable for volatile and evolving decentralized systems. This description details how the model derives the final trust value for each individual calculable feature. Following this step, the system then aggregates these individual feature trust scores to form an overall trust estimate for an entity. This aggregation acts akin to the Deterministic Algorithm, where the final entity trust score is calculated as a weighted average of the outputs from the probabilistic algorithm for each feature. Similar to the deterministic approach, the definition of optimal weights for this final aggregation can be achieved through various methods, including expert knowledge, empirical testing, or machine learning optimization. Furthermore, the parameters within the probabilistic model itself (e.g., ‘base lambda‘, ‘growth rate‘, ‘uncertainty penalty‘) can also be finetuned to achieve the most suitable scenario for a given use case.

# 5. Identity and Role management system

## 5.1 Definition of identities and roles in Swarmchestrate

Swarmchestrate is a decentralized orchestration framework designed to manage resources and workloads seamlessly across the Cloud-to-Edge continuum. To ensure trust and security among its participants who may operate in different administrative domains it adopts a Self-Sovereign Identity (SSI) approach for establishing and managing identities. Each participant is given a unique identity that anchors core attributes to a Decentralized Identifier (DID).

Within Swarmchestrate, five main roles are defined:

1. Application Owner
    - Submits resource requirements and workload specifications (for instance, TOSCA-based descriptions).
    - Needs verifiable assurance about which providers can be trusted to host applications securely and reliably.
2. Capacity Provider
    - Owns or controls resources in the Cloud-to-Edge continuum (e.g., servers, edge devices, cloud instances).
    - Registers a DID on-chain and issues Verifiable Credentials (VCs) that describe its resource capabilities (such as CPU, memory, storage).
    - Maintains an on-chain reputation score, which reflects historical performance and reliability.
3. Resource Agent
    - Discovers and verifies available resources, using the VCs issued by Capacity Providers.
    - Schedules and allocates tasks to resources based on a combination of factors such as capacity, location, latency, and the provider's reputation.
    - Relies on the on-chain data (DIDs, revocation entries, and reputation scores) for resource selection.
4. System/User Interface
    - Presents a consolidated gateway for Application Owners and Capacity Providers to interact with the Swarmchestrate environment.
    - Facilitates registration, credential issuance, and retrieval of on-chain metadata, ensuring a user-friendly experience.
5. Universe

- Represents a top-level conceptual domain or trust boundary in Swarmchestrate. It represents the recognized ecosystem where application components will be deployed and managed by Swarmchestrate.
- Holds a unique public key on-chain, which grants it authority to update or validate key governance actions, such as rating providers after successful or failed deployments.
- Ensures that any changes to on-chain reputations or revocation records are legitimate.

These roles collectively enable decentralized collaboration without reliance on a single authority. Each participant's identity, anchored by a DID, is verifiable on-chain, providing a robust foundation for trust and secure orchestration.

Why capacity providers register a DID and issue VCs

- Recording a Decentralized Identifier (DID) on-chain gives each provider a permanent, tamper-proof identity.
- The capacity's CPU, RAM and storage are published in Verifiable Credentials (VCs) signed with the same key as the DID. Keeping these credentials off-chain protects operational details from public view.

Benefits of the approach

1. Eliminates dependence on a central authority.
2. Ensures sensitive data never appears on the blockchain.
3. Allows providers to update individual resource claims (by issuing new credentials off-chain without disclosing extra information).

Why reputation is tracked on-chain and how the 0-10 score is calculated

- Swarmchestrate sends work to machines that don't necessarily trust one another. To pick good hosts we look at how well each one has behaved before.
- We store that track record right on the blockchain. Once it's written nobody can change or delete it. Everyone can see the same history.
- Each host gets a score from 0 to 10. The range is small, so it's cheap and blockchain-efficient to save, yet big enough to show who's dependable and reliable.

Figure 5 illustrates how these roles interact within a Swarmchestrate "Universe." First, the Universe itself registers a public key on-chain (1a), effectively establishing a trust domain. Capacity Providers subsequently register their DIDs on-chain and publish capacities using off-chain logic, accompanied by Verifiable Credentials (2). Application Owners then submit workload requirements (3), which the Resource Agents must fulfil by locating and verifying appropriate capacities (4–5).

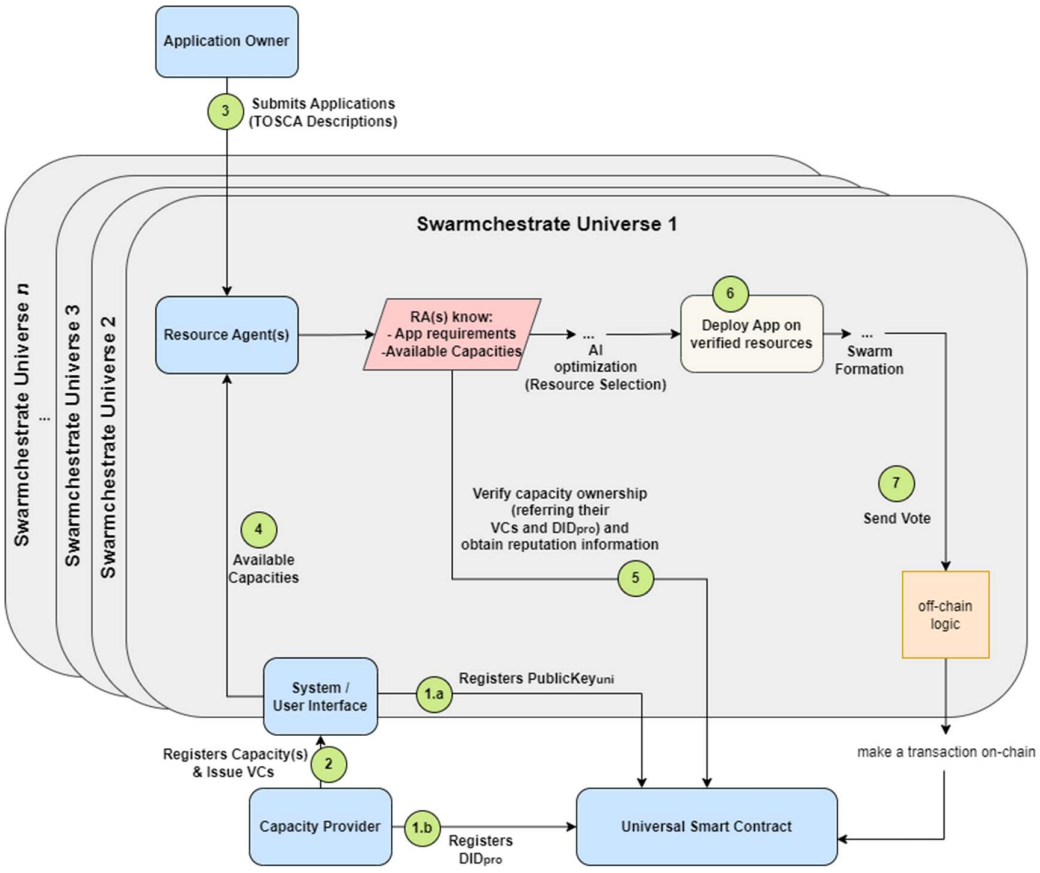



*Figure 5 The proposed Smart-Contract-based Identity Management framework*

During verification, a Resource Agent checks the provider's DID and any relevant VCs. It queries on-chain data to confirm that the provider's DID is valid, that no revocation entries exist, and to view the provider's current reputation score. Once resources are selected, the Resource Agent deploys tasks onto those verified nodes (6). Finally, off-chain performance monitoring may trigger the Universe to update a Capacity Provider's reputation score (7), submitting any changes on-chain.

On-chain vs. off-chain in Swarmchestrate.

The DID and Role management system for Swarmchestrate keeps the heavy data off the blockchain and stores only what really needs to be there. The Universal Smart Contract holds three things: an unchangeable ID for each provider (DID), any proofs that a credential was revoked (credential hash) and a 0 to 10 reputation score. Everything bulky or sensitive stays outside of the blockchain, off-chain. Posting the vote on-chain updates the provider's reputation in one atomic step. Anyone can then trust the score without seeing the raw metrics, keeping both costs and privacy concerns under control.

## 5.2 Decentralised identifiers, Verifiable credentials and credentials management

Swarmchestrate's identity management strategy is grounded in SSI principles, leveraging DIDs and VCs to establish decentralized trust and protect user privacy. Rather than storing all identity data on a central server, the framework uses blockchain technology to anchor only minimal critical metadata on-chain, such as DID references, revocation proofs, and aggregated reputation scores.

1. Decentralized Identifiers (DIDs)
    - Each Resource Provider generates their own DID. This DID includes cryptographic keys and a DID Document.
    - The DID itself is registered on-chain (in a universal smart contract), ensuring that key references and ownership proofs are tamper-proof.
    - DID Documents are structured so that core identity details (like public keys) remain immutable, while mutable fields (such as reputation or revocation status) can be updated over time.
2. Verifiable Credentials (VCs)
    - Capacity Providers issue VCs describing resource attributes (e.g., number of CPU cores, available memory, performance benchmarks).
    - These credentials are signed off-chain using the issuer's private key and can be verified by any Resource Agent against the issuer's DID.
3. Credential Revocation
    - A Capacity Provider can revoke an issued credential by placing its hash in an on-chain revocation registry. This registry is part of the universal smart contract's data structure, conceptually aligning with a three-dimensional distributed ledger model.
    - Whenever a credential is presented, the verifier checks the ledger to confirm that its hash has not been flagged as revoked. If it appears in the registry, the credential is considered invalid.
4. Off-Chain Verification Flow
    - Swarmchestrate avoids storing large or private data on-chain. All sensitive credential details remain off-chain, secured in holders' wallets.
    - Verifiers *(Resource Agents)* perform signature checks and consult the on-chain revocation mapping to confirm that the credential remains valid.
    - This approach optimizes scalability and reduces on-chain costs, while still providing decentralized trust guarantees.

By combining these SSI-based mechanisms, Swarmchestrate ensures robust identity management. The framework enforces integrity through the blockchain's immutability and the cryptographic safeguards of DIDs and VCs, all while preserving user autonomy and minimizing reliance on central authorities.

Figure 6 illustrates how these components interact via a multi-dimensional ledger approach. Each Universe *(shown as PUB_uni_i)* has its own domain of trust, carrying a corresponding on-chain wallet address. This design enables multiple universes to coexist, while Capacity Providers register their DIDs *(DID_pro_i)* above each universe layer. The ledger captures both the immutable core of a DID *(e.g., public keys)* and the mutable fields *(e.g., reputation, revocation status)*, ensuring that new entries do not overwrite historical states.

At the top, the Universal Smart Contract oversees critical operations such as DID registration, credential revocation, and aggregated reputation updates. In doing so, Swarmchestrate combines the immutability of blockchain with the flexibility of off-chain VCs, enabling privacy-preserving identity flows that still maintain provable integrity and trust.

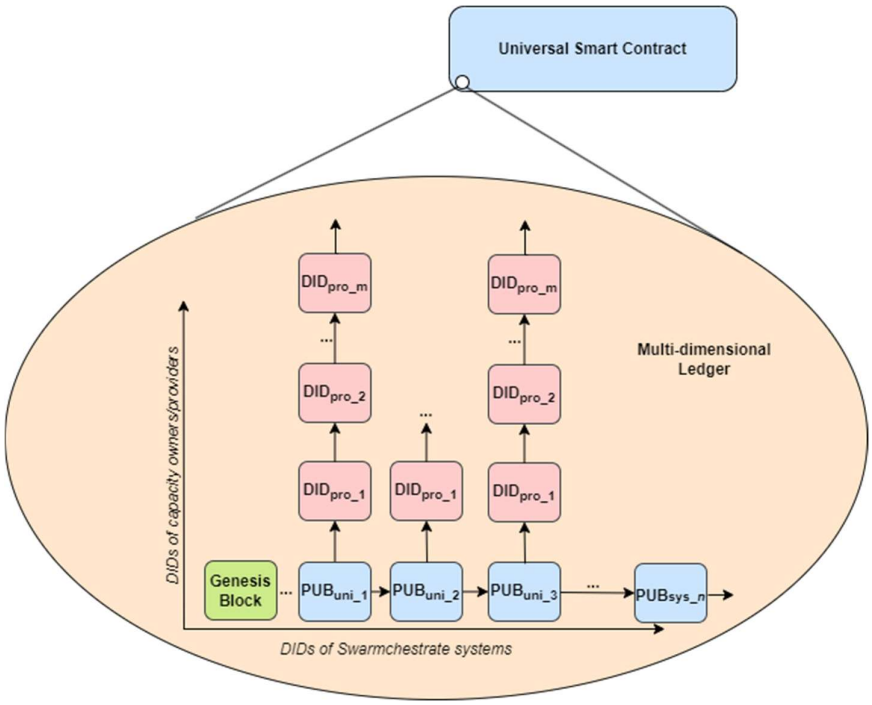



*Figure 6 Conceptual overview of the 3DDL ledger structure.*

## 5.3 Initial system design

The Swarmchestrate identity solution brings together on-chain smart contracts, off-chain credential exchange, and a reputation mechanism in a cohesive design:

1. Universal Smart Contract
    - Maintains a mapping of registered DIDs, each associated with essential identity data.

- Records the credentials' revocation hashes in a dedicated registry, preventing tampering or erasure of historical data.
- Facilitates an on-chain reputation system, allowing authorized entities (the Universe) to submit updates after resource deployments are evaluated.

2. Three-Dimensional Ledger Structure
   - Swarmchestrate logically separates:
     - Universes (each with a unique public key and trust domain).
     - DID Records (split into immutable and mutable parts).
     - Revocation Entries (a chain of revoked credential hashes).
   - Although physically stored in one smart contract for simplicity, these logical partitions promote better organization and auditing.
3. Reputation Model
   - Each Capacity Provider has an on-chain reputation score, typically normalized to a 0–10 range.
   - Scores update whenever the Universe posts a "vote" transaction that reflects the success or failure of a resource deployment.
   - Because only the Universe's key is authorized to submit these updates, it establishes a secure and auditable way to manage trust data without arbitrary tampering.
4. Interaction Flow
   - **Registration:** A new participant generates a DID and registers it on-chain.
   - **Credential Issuance:** A Capacity Provider issues a signed VC (off-chain) describing its resources, referencing the DID.
   - **Verification:** Resource Agents consult the provider's DID on-chain, confirm the VC's signature, and check for revocation before scheduling tasks.
   - **Reputation Update:** If the provider's resources fulfil or fail their orchestration obligations, the Universe submits a transaction that updates the provider's reputation on-chain.

Figure 7 depicts the earliest steps in this process:

- A Universe entity generates its public key (Pub_Uni) and registers it in the Universal Smart Contract, establishing the first dimension of trust.
- A Capacity Provider then generates a DID (DID_Prov) and registers it on-chain, entering the second dimension.

- Each registration is confirmed by the Smart Contract, ensuring a verifiable, tamper-proof record of newly introduced entities.

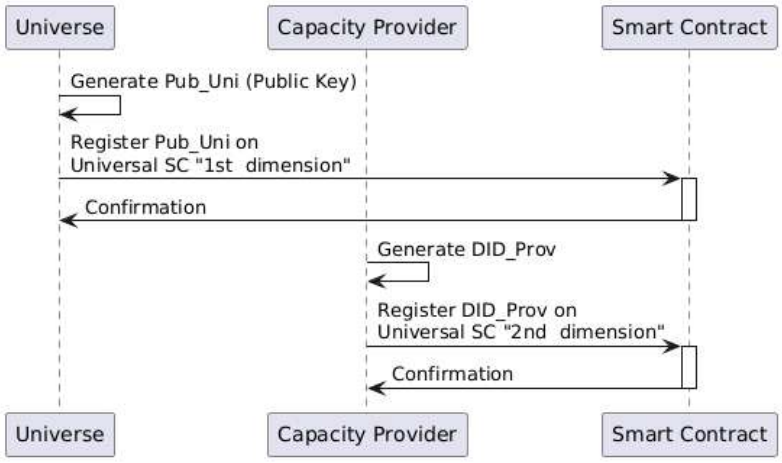



*Figure 7 Registration Flow*

Figure 8 illustrates how, after registration, a Capacity Provider can issue VCs describing the attributes of its resources. These credentials remain off-chain but can be queried by Resource Agents, which verify them against the on-chain DID and reputation data. Post-deployment, the Monitoring Component collects resource usage metrics and shares relevant information off-chain. If the resource under- or over-performs, the Universe submits a trust "vote" transaction to the Smart Contract, updating the provider's reputation on-chain.

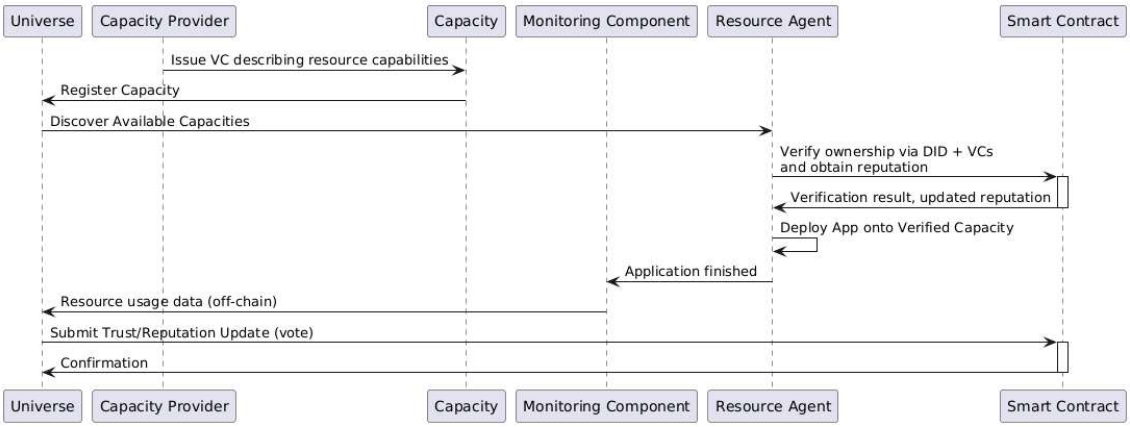



*Figure 8 VC Issuance and Verification*

In this way, Swarmchestrate provides a foundational model for secure, transparent, and decentralized orchestration, where trust naturally emerges from verifiable credentials and on-chain reputation records, and revocation or identity updates can be carried out with minimal overhead.

## 5.4 Alignment with the Project Wide Trust Model

From section 3.3.2 and 3.3.4 of this deliverable, we can visualize a **three-pillar trust framework** for Swarmchestrate:

1. **Identity trust:** cryptographic proof of stakeholders' identities using DIDs and VCs. [Section 3.3.4]
2. **Behavioral trust:** reputation collected from real time monitoring and past performance. [Section 3.3.4]
3. **Contextual trust:** factors tied to deployment conditions and to the boundaries between and within swarms. [Section 3.3.2]

The table below summarizes the contribution of the SSI and Identity and Role Management layer to the project-wide trust model:

| PILLAR | Contribution of the SSI / Identity and Role Management component |
|---|---|
| Identity trust | Generates and resolves DIDs on chain, issues and verifies VCs, stores minimal metadata on-chain. |
| Behavioral trust | Maintains an on-chain reputation score for every Capacity Provider, the Universe key is the only authority that updates scores after monitoring. |
| Contextual trust | Defines intra- and inter-swarm boundaries with Universe public keys, establishing clear identity-based separations. |

# 6. Fair and Private Resource Ranking in Swarmchestrate

This section outlines the resource selection mechanism in Swarmchestrate, which is designed to identify the most suitable set of computational resources based on the application's Quality of Service (QoS) requirements, as specified in the TOSCA service template.

The ranking process operates on three main inputs: the available resource offers, the QoS goals defined by the application, and dynamically retrieved reliability metrics associated with each offer. Reliability in this context reflects a holistic assessment of the operational characteristics of a resource that may influence its ability to meet QoS targets. These characteristics include the historical frequency of failures, general availability, accuracy of declared resource specifications, observed performance patterns, and network accessibility under realistic conditions. Importantly, these reliability indicators contribute directly to the trust assessment process, where each characteristic serves as a calculable feature within the probabilistic trust model. The dynamic trust evaluation not only quantifies confidence in a resource's ability to deliver consistent performance but also ties this evaluation to the entity's identity, as defined by its verifiable credentials and historical behavior. This enables an adaptive understanding of trust—one that can evolve over time and reflect both the technical and behavioral aspects of entities across decentralized Swarmchestrate environment.

An AI- or optimization-based ranking algorithm processes these inputs to derive a prioritized list of resource candidates. The goal is to identify the resource configuration that not only satisfies the declared application needs but also maintains high standards of operational integrity and predictability. The overall workflow is illustrated in Figure 9.

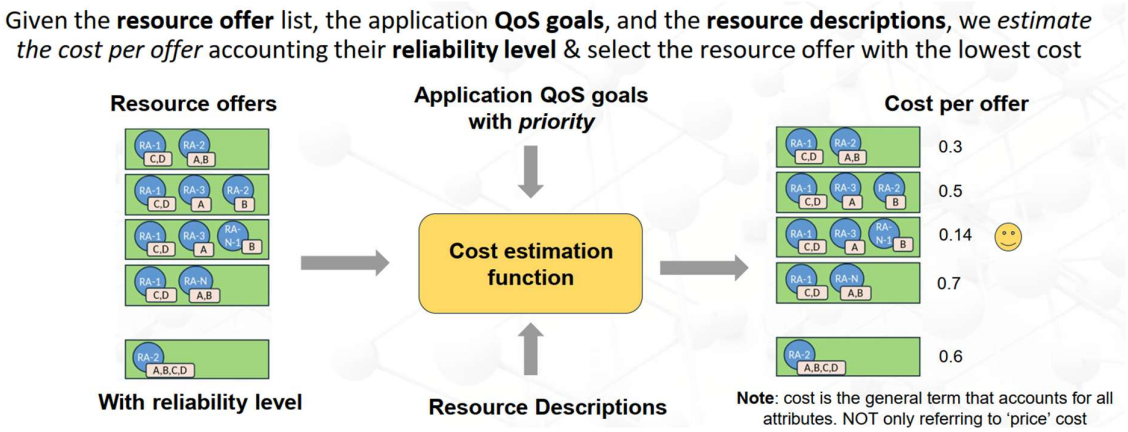



*Figure 9. Resource Ranking in Swarmchestrate (Figure from AI working group).*

### Trust and Security Concerns in Resource Ranking.

A key challenge arises in preserving the confidentiality of sensitive resource attributes—such as hardware capabilities, processing speed, or latency characteristics—while still enabling effective decision-making in resource ranking. In the current setup, such descriptors are disclosed to the lead Resource Agent (RA), introducing several critical risks.

First, this exposure may lead to conflicts with jurisdictional constraints or ethical obligations surrounding data protection, particularly when the RA operates across regulatory boundaries. Second, an untrusted or compromised RA could unfairly influence the ranking outcome by manipulating inputs, prioritizing specific resources, or harvesting system-level insights that were not intended to be shared. Finally, open access to detailed resource profiles increases the surface for unauthorized data disclosure.

These concerns underscore the need for a privacy-preserving ranking framework that ensures correctness and fairness without requiring raw access to sensitive information. In this context, Swarmchestrate aims to integrate secure-by-design methodologies that uphold data confidentiality while enabling reliable, transparent, and verifiable resource matchmaking.

## 6.1 Private Resource Ranking in Swarmchestrate

To address these concerns, a private ranking mechanism based on verifiable Functional Encryption (FE) has been developed. The proposed solution allows ranking operations to be performed on encrypted data while ensuring verifiability of results.

Key Aspects:

- **Encrypted Resource Descriptions**: All criteria required for ranking (e.g., CPU, latency, reliability) are encrypted using FE.
- **Privacy-Preserving Ranking Function**: The lead RA computes the ranking based on encrypted values without accessing the raw resource descriptions.
- **Verifiability**: The system ensures that the returned ranking is correct and has not been manipulated.
- **Fairness and Transparency**: All participating RAs can verify that the ranking was conducted fairly.

The implementation utilizes an Inner Product Functional Encryption (IPFE) scheme, allowing the ranking function to compute an inner product on encrypted values while ensuring verifiable decryption. The FE mechanism prevents the lead RA from learning individual resource descriptions, mitigating risks of bias or data exposure. A high-level overview of the solution is outlined in Figure 10.

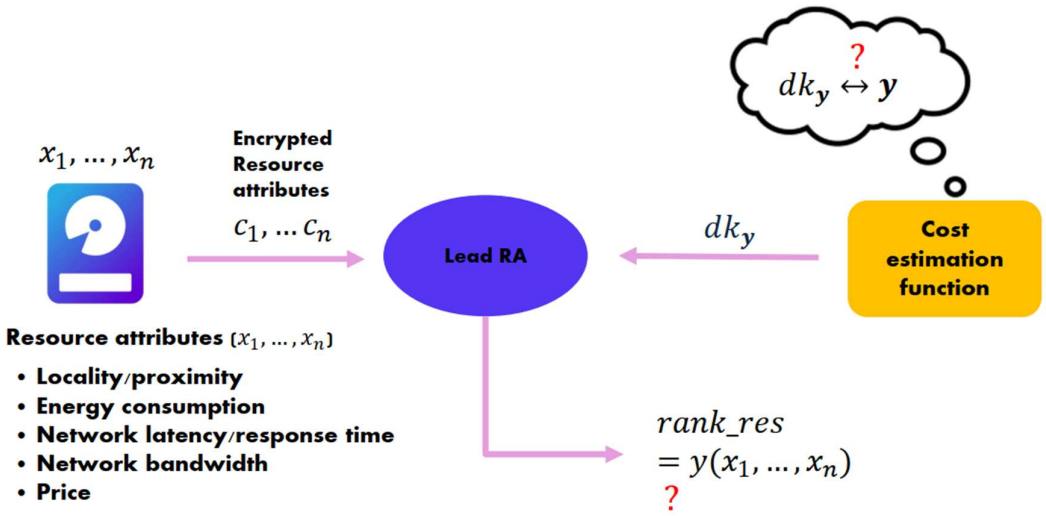



*Figure 10: High-level overview of fair Resource Ranking solution: (i) all criteria that are required for ranking resources (like hardware specifications, CPU, latency, …) is encrypted using FE, (ii) The ranking function is run on the encrypted data, (iii) The m machine that is running the ranking process cannot maliciously reorder and manipulate the scoring output, (iv) Everyone can verify the returned score is computed correctly ensuring fair, transparent, and verifiable ranking process.*

## 6.2 Initial system design

Let $f$ denote the cost estimation function that takes as input the set of resource offers $O$, a vector of QoS resource descriptions $Q$, and a priority vector $P$ associated with QoS attributes. Specifically, we have $f(O, Q, P)$,

where, for each $o \in O$,

$$P = (p_1, p_2, \ldots, p_n)$$
$$Q = (f_{q_1}(o), f_{q_2}(o), \ldots, f_{q_n}(o))$$

The cost function can be expressed as a weighted sum of the individual QoS evaluations. For $o \in O$, we have:

$$f(o, Q, P) = p_1 f_{q_1}(o) + p_2 f_{q_2}(o) + \cdots + p_n f_{q_n}(o)$$

This is equivalent to the inner product of vectors $P$ and $Q$:

$$f = \langle P, Q \rangle = P^T Q$$

**Privacy-Preserving Evaluation via Functional Encryption:** Based on the previous discussion, suppose $P$ (QoS priority) is public and $Q$ (QoS attributes) is private. To execute this function in a privacy-preserving manner, we propose the use of inner product functional encryption (IPFE). This enables secure computation of the inner product $\langle P, Q \rangle$ without revealing the private vector $Q$.

**Remark 1. Dual-Mode Ranking Support:** To accommodate varying privacy preferences, a dual-mode ranking mechanism can be implemented. This allows resource owners to choose between a privacy-preserving mode (using encrypted QoS attributes) and a standard non-private mode (where attributes are shared in plaintext). The ranking algorithm processes all available resources accordingly, and the final output reflects a unified list that includes both private and non-private entries. This flexible approach empowers resource owners to select the level of confidentiality that best suits their needs, offering a more inclusive and customizable resource selection experience within Swarmchestrate.

**Remark 2. Private Ranking Mechanism for Borda Voting Cost Function:** Here, we propose a privacy-preserving ranking mechanism tailored for the Borda voting cost function introduced in D1.1, as it aligns well with our IPFE techniques proposed in [17]. In this scheme, each QoS attribute is ranked independently—e.g., higher bandwidth is ranked better (descending order), while lower latency is preferred (ascending order). Offers receive scores based on their rank within each attribute, with ties sharing the top score for the tied position. These scores are then aggregated using the publicly known attribute priorities $P$, and optionally adjusted with reliability factors (either additively or multiplicatively). Since the Borda scoring process relies on structured and separable computations across attributes, it maps naturally onto the IPFE-based setting, as it avoids the need for normalized floating-point representations.

### 6.2.1 Preliminaries

*Notation*

For $m \in \mathbb{N}\text{^} *, [m]$ is the set $\{1, \ldots, m\}$. Vectors are denoted in bold lowercase letters. The inner product between two vectors $x = (x_1, \ldots, x_n)$ and $y = (y_1, \ldots, y_n)$ is $\langle x, y \rangle = x_1 y_1 + \ldots + x_n y_n$ and their Hadamard product is $x \circ y = (x_1 y_1, \ldots, x_n y_n)$. For a set $X$, we use $x \leftarrow \$ X$ if $x$ is sampled uniformly at random from $X$ and $x \leftarrow D_{\{X,\sigma\}}$ if x is sampled from the standard Gaussian distribution in $X$ for deviation $\sigma$. A PPT adversary $SDV$ is a randomized algorithm for which there exists a polynomial $P(x)$ such that for all input $x$, the running time of $SDV(x)$ is bounded by $|P(x)|$. A function $f : \mathbb{N} \mapsto \mathbb{R}$ is said negligible if $\forall\, c \in \mathbb{N}, \exists\, x_0 \in \mathbb{N}$ such that $\forall\, x \geq x_0, f(x) < x^{\{-c\}}$. We denote $negl(\cdot)$ an arbitrary negligible function.

*Inner Product Functional Encryption*

An Inner Product Functional Encryption (IPFE) scheme enables users to calculate the inner product between a vector of plaintexts and a vector for which they possess the functional decryption key, without exposing the plaintexts.

**Definition 1 [Inner Product Functional Encryption (IPFE)]**

An IPFE scheme for a message space M and a vector space Y is a tuple IPFE = (Setup, Enc, KeyGen, Dec) such that:

- $Setup(1^\lambda)$: The setup algorithm Setup is a probabilistic algorithm that on input the security parameter $\lambda$, outputs a master public and private key pair $(mpk, msk)$.

- $Enc(mpk, x)$: The encryption algorithm Enc is a probabilistic algorithm that on input the master public key mpk and a message $x = (x_1, \dots, x_n) \in M$, outputs a ciphertext $c$.
- $KeyGen(msk, y)$: The key generation algorithm KeyGen is a deterministic algorithm that on input the master secret key msk and a vector $y \in Y$, outputs a functional decryption key $dk_y$.
- $Dec(dk_y, c)$: The decryption algorithm Dec is a deterministic algorithm that on the input of a functional decryption key $dk_y$ and a ciphertext $c$, outputs the inner product of $y$ on the plaintexts $\langle x, y \rangle$.

Note that that the decryption key in **Definition 1** depends only on the vector y and not on any specific ciphertext. This means that anyone with access to the functional decryption key $dk_y$ can retrieve the quantity $\langle x, y \rangle$ for any message $x$.

### 6.2.2 Multi-Client Inner Product Functional Encryption

The notion of Multi-Client Inner Product Functional Encryption (MCIPFE) originally comes from the need of users to encrypt their data separately, hence the name *multi-client*. Different clients could be registered in the same system but encrypt their data under their own encryption key, or label.

**Definition 2 [Multi-Client Inner Product Functional Encryption]**

An MCIPFE scheme for a message space $M$ and a vector space $Y$ is a tuple $MCIPFE = (Setup, Enc, KeyGen, Dec)$ such that:

- $Setup(1^\lambda)$: The setup algorithm Setup is a probabilistic algorithm that on input the security parameter $\lambda$, outputs a master public and private key pair $(mpk, msk)$.
- $Enc(mpk, x, \ell)$: The encryption algorithm Enc is a probabilistic algorithm that on input the master public key $mpk$, a message $x = (x_1, \dots, x_n) \in M$ and a label $\ell$, outputs a ciphertext $c^\ell$.
- $KeyGen(msk, y, \ell)$: The key generation algorithm $KeyGen$ is a deterministic algorithm that on input the master secret key $msk$, a vector $y \in Y$ and a label $\ell$, outputs a functional decryption key $dk_y$.
- $Dec(dk_y^\ell, c^{\ell'})$: The decryption algorithm Dec is a probabilistic algorithm that on input a functional decryption key $dk_y^\ell$ and a ciphertext $c^{\ell'}$, outputs the inner product $\langle x, y \rangle$ if $\ell = \ell'$, and $\perp$ otherwise.

### 6.2.3 Verifiable Functional Encryption

FE was first introduced by Boneh et al. in [18] as a generalization of Public-Key Encryption (PKE). Since then, numerous new definitions and different approaches have been proposed [19], [20], [21]. As its name suggests, functional encryption schemes are usually implemented with a limited set of functions for a specific use case. In our work, we focus on Inner Product Functional Encryption (IPFE), in which the function is the inner product for a given vector. Prior to our work, numerous IPFE protocols have been proposed, based on different hardness assumptions. Among these, the learning with error (LWE) problem has been very popular [22],

[23], [24] for its security against quantum attacks. However, the structure of the ciphertexts makes it difficult to design an efficient verification method, and none of these cryptosystems has been endowed with such a feature. Another popular approach is the use of the Decisional Diffie-Hellman problem (DDH) [25], [24], [26] or related assumptions. The interest of these constructions is the efficiency and simplicity of their algorithms. Moreover, the ciphertexts being elements of a cyclic group, designing a verification feature based on zero-knowledge proof (ZKP) on a discrete logarithm is extremely practical.

Meanwhile, several recent works address the problem of verifiable decryption in FE and propose generic methods. Among them, we cite the work of Badrinarayanan et al. [25] that formally defines for the first time the need for a verification method for FE. They provide a method of constructing a verifiable protocol from any FE scheme, including IPFE. However, this construction is primarily theoretical and lacks efficiency in terms of real use-case application. Subsequently, recent protocols have introduced a more pragmatic approach [27], [28] showing that verifiable IPFE can also be practical. However, the use cases they consider primarily focus on malicious users. This differs from our approach. As this work addresses the trust issues between two parties, it is important to mention the existence of decentralized IPFE [29], [26]. In a decentralized setup, data holders do not rely on a trusted authority for key management, which solves the privacy issues of classical IPFE. This field has been widely studied in the past few years and, despite being more resource-demanding than previous schemes, some practical works have been designed. However, this setup requires constant online participation from the users, as well as being more computationally expensive.

We use verifiable FE in a real use case scenario, namely achieving fairness in private scoring. We chose to work on the centralized construction of Castagnos et al. [10], which has the advantage of proposing a protocol with efficient decryption, hence more suitable for our use case.

## 6.3 System and Threat Model

***System Model***: The system model we consider consists of three entities: (i) Key Curator (C), (ii) Users (U), (iii) Analyst (A).

- Key Curator (C): We assume the existence of an authority C, which is responsible for the Setup phase and the generation of the decryption keys upon request of an analyst for a specific vector.
- Users (U): Let $U = u_1, \ldots, u_n$ be a set of users holding sensitive data. They each encrypt their data under the public parameters generated by the curator and send them to the latter. The n users involved in the system individually encrypt their sensitive data into ciphertext using the public parameters generated by C. Then, they send their encrypted data to the curator.
- Analyst (A): We consider an external data analyst A who expects the result of selective computation on the data held by the users. A interacts with the curator and sends functional queries to C.

In the context of Swarmchestrate, the users (U) correspond to the resource providers who provide QoS attribute vectors in encrypted form, while the analyst (A) represents the lead resource agent (RA) responsible for evaluating rankings based on specific QoS priorities. Also, message $x = (x_1, \ldots, x_n) \in M$ corresponds to private vector $Q$ of QoS attributes.

***Threat Model***: The curator C generates the master key through the Setup phase and administers the data collection infrastructure. We do not require C to be trusted by an analyst A and consider the possibility that C sends the wrong decryption key to mislead the results of the analyst or that he does not send the data at all during the transaction. On the user side, we reasonably consider that the users are honest in the data they provide, but they are not compelled to trust each other.

In terms of security, we assume that the curator does not collude with the analyst, as this would provide the analyst with access to all user data. We highlight again that users trust the curator with their data, as they have already been rewarded for them. Furthermore, this is required in a classical IPFE setup: given the capability to generate a decryption key for arbitrary vectors, C can compute trivial inner products on the ciphertexts, hence recovering the plain data. The construction of IPFE schemes that do not rely on trusted authority requires a decentralized setup. In this type of construction, the users themselves generate the decryption keys, which implies constant online involvement.

## 6.4 Constructing a Verifiable MCIPFE

Here, we employ cryptographic techniques developed in our earlier work in [1]. Consider a scenario where a data analyst wants to perform selective computations on sensitive data $x = (x_1, \ldots, x_n)$ held by a single user. In this scenario, a key curator runs the Setup algorithm, sets mpk as public information, and keeps msk secret. The user encrypts the message x into a ciphertext $c = (c_1, \ldots c_n) \leftarrow$ Encrypt(mpk, x). The analyst, who has access to c, wants to calculate $\langle y, x \rangle$ for a vector y. Therefore, she sends a request to the curator to obtain the functional decryption key $dk_y$. The curator, given the analyst's vector y and secret key msk, produces the decryption key $dk_y$ through KeyGen and sends it back to the analyst. Finally, the latter can use Decrypt to obtain the result.

In this scenario, we require the decryption key $dk_y$ to be specific to a ciphertext or set of ciphertexts. This restriction permits better access control and is recommended in the use case we consider, where an analyst purchases specific results. This can be achieved using an MCIPFE construction that we defined in **Definition 2**.

Further down, we begin with the basic scheme of Castagnos et al. [28] and note its limitations and security concerns for our system model. Then we extend it by adding a multi-client feature, multi-user support and we consider verifiable decryption.

### 6.4.1 Construction of Castagnos et al.

To present our contribution, we first recall the IPFE scheme of Castagnos et al. in Protocol 1. Mark that the system model supported by this protocol admits a single user.

In previous constructions based on Additive-ElGamal encryption scheme, the decryption procedure relies on computing a discrete logarithm in a cyclic group. This problem is known to be hard in the general case, hence the correctness of the protocols requires length restrictions on the exponent size. In terms of practical use, this appears to be a serious issue, as it requires strict limitations on the size of the messages as well as the set of decryption keys that can be delivered to analysts. The IPFE construction presented in this section relies on a DDH group with an easy DL subgroup. In short, it consists in generating a cyclic group G together with a subgroup F of G such that the discrete logarithm is easy to compute in F but hard in G.

Protocol 1 [IPFE over $\mathbb{Z}$ under the DDH-f assumption] [10]

Following **Definition 1** , this protocol consists of 4 algorithms described in Figure 11.

For the detailed description of the algorithmic pair (Gen, Solve), we refer the reader to the definition of the Decisional Diffie-Hellman with an Easy Subgroup problem as presented in [17]. Moreover, the selection of the deviation parameter σ is analyzed in the context of the security guarantees, also discussed in [26].

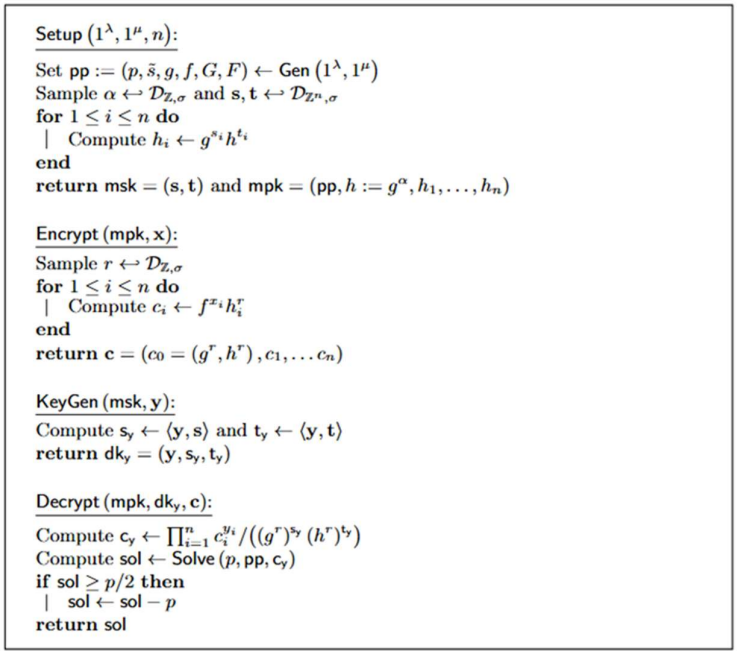



*Figure 11 Castagnos et al. IPFE algorithms.*

***Limitations***: We motivate the construction of **Protocol 2** by highlighting several limitations of the previous construction concerning the properties expected by our system model. First, the protocol of Castagnos et al. is not multi-client, i.e., the functional decryption key generated by the curator is not specific to a ciphertext, hence giving the possibility to the data analyst to get

more information than the one she originally requested. Secondly, this scheme has been designed for a set of linearly independent requested vectors small enough not to endanger the security of the plaintexts, as detailed below. The set of a functional data analyst is much bigger in a MCIPFE scheme, due to label dependency. However, allowing the generation of key brings forth a different security issues on the secret key itself themselves. Finally, this protocol only supports a single user. Hence, we design a variant supporting multi-user support for n users encrypting data independently.

***Multi-Client Support***: The multi-client construction presented in **Definition 2** allows a user to encrypt messages through different users or labels and divide the user data into a set of encrypted data into smaller subsets. In **Protocol 2** this allows us to add a dependency between the decryption key and the ciphertext. Therefore, an analyst holding a decryption key $\{dk_y\}^{\{(\ell)\}} \leftarrow KeyGen(msk, y, \ell)$ can be calculated ⟨y, x⟩ for a decryption key x for a functional decryption key y encrypted under the same label l. Consecutively, to restrict the access of a data analyst to a set of data, it is necessary to change the label of further encrypted data. In particular, if the curator wishes to sell a single computation, it is possible to deliver a functional decryption key specific to a single ciphertext encrypted with a specific label.

### 6.4.2 Adding Multi-Client Support

Now, we introduce our corresponding MCIPFE construction following **Definition 1** We propose a new decryption key generation allowing the curator to output a functional decryption key that depends on a specific set of ciphertexts, encrypted under a common label. First of all, this construction requires a preliminary encoding of this label.

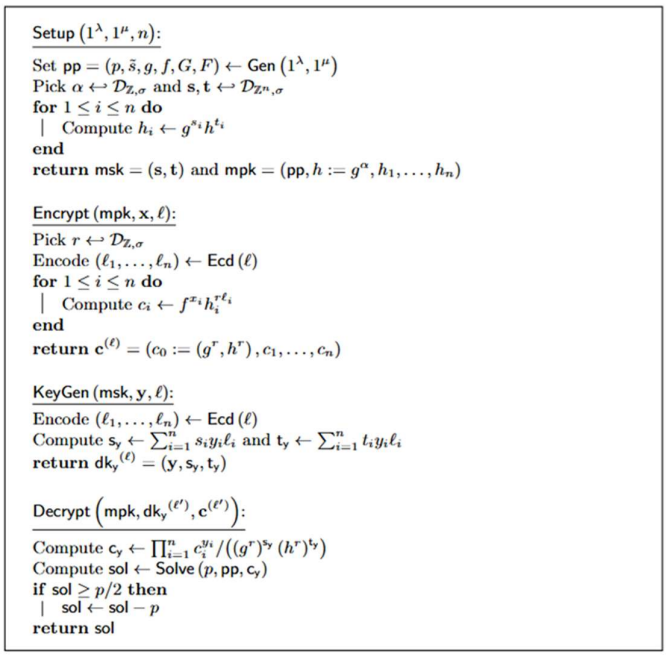
***Label Encoding***: To construct an MCIPFE, we design an encryption and decryption procedure that allows a user to encrypt a ciphertext under a specific label $\ell \in \mathbb{N}$. To ensure this specificity, hence the security of the protocol, it is necessary to verify that, given a decryption key for a label $\ell$, an adversary cannot forge a decryption key for another label $\ell'$. As the decryption key from Protocol 1 is an inner product, the multi-client decryption key cannot be linear in $\ell$. Therefore, we consider an encoding method $Ecd : \mathbb{Z} \rightarrow \mathbb{Z}^n; \ell \mapsto (\ell_1, \dots, \ell_n)$ where each $\ell_i$ depends non-linearly on $\ell$. This idea has already been proposed by Abdalla *et al.* [6] for the general case using a *pseudo-random function* PRF(i, $\ell$) as a basis for their encoding. However, this general approach is not the best choice for our protocol which relies on a public encoding. Therefore, we believe that the use of a hash function would be sufficient to guarantee the non-linearity of the encoding and the unforgeability of a label $\ell$, while providing better performances and better flexibility on the choice of labels.

Specification: we use a simple first and second pre-image resistant cryptographic hash function $H : \{0,1\}^* \rightarrow \mathbb{Z}_{\{n \cdot p\}}$. The output of the hash h can be split afterwards into n components of range p with a canonical projection $\pi : h := h_{\{n-1\}} || \dots || h_0 \mapsto (h_{\{n-1\}}, \dots, h_0)$. Eventually, given as input a label $\ell$, we define the following encoding algorithm:

$$\begin{aligned} \text{Ecd} : \{0,1\}^* &\rightarrow \mathbb{Z}_{n \cdot p} \rightarrow \mathbb{Z}_p^n \\ \ell &\mapsto H(\ell) \mapsto (\ell_1, \dots, \ell_n) \end{aligned}$$

**Protocol 2 [MCIPFE scheme under the DDH-f assumption]**

Our multi-client variant of Protocol 1 consists of 4 algorithms (Setup, Encrypt, KeyGen, Decrypt) described in Figure 12.



**Setup** $(1^\lambda, 1^\mu, n)$:

Set $\mathsf{pp} = (p, \tilde{s}, g, f, G, F) \leftarrow \mathsf{Gen}\,(1^\lambda, 1^\mu)$
Pick $\alpha \hookleftarrow \mathcal{D}_{\mathbb{Z},\sigma}$ and $\mathbf{s}, \mathbf{t} \hookleftarrow \mathcal{D}_{\mathbb{Z}^n,\sigma}$
**for** $1 \le i \le n$ **do**
  Compute $h_i \leftarrow g^{s_i} h^{t_i}$
**end**
**return** $\mathsf{msk} = (\mathbf{s}, \mathbf{t})$ and $\mathsf{mpk} = (\mathsf{pp}, h := g^\alpha, h_1, \ldots, h_n)$

**Encrypt** $(\mathsf{mpk}, \mathbf{x}, \ell)$:

Pick $r \hookleftarrow \mathcal{D}_{\mathbb{Z},\sigma}$
Encode $(\ell_1, \ldots, \ell_n) \leftarrow \mathsf{Ecd}\,(\ell)$
**for** $1 \le i \le n$ **do**
  Compute $c_i \leftarrow f^{x_i} h_i^{r\ell_i}$
**end**
**return** $\mathbf{c}^{(\ell)} = (c_0 := (g^r, h^r), c_1, \ldots, c_n)$

**KeyGen** $(\mathsf{msk}, \mathbf{y}, \ell)$:

Encode $(\ell_1, \ldots, \ell_n) \leftarrow \mathsf{Ecd}\,(\ell)$
Compute $\mathsf{s_y} \leftarrow \sum_{i=1}^n s_i y_i \ell_i$ and $\mathsf{t_y} \leftarrow \sum_{i=1}^n t_i y_i \ell_i$
**return** $\mathsf{dk_y}^{(\ell)} = (\mathbf{y}, \mathsf{s_y}, \mathsf{t_y})$

**Decrypt** $\left(\mathsf{mpk}, \mathsf{dk_y}^{(\ell')}, \mathbf{c}^{(\ell')}\right)$:

Compute $\mathsf{c_y} \leftarrow \prod_{i=1}^n c_i^{y_i} / \left((g^r)^{\mathsf{s_y}} (h^r)^{\mathsf{t_y}}\right)$
Compute $\mathsf{sol} \leftarrow \mathsf{Solve}\,(p, \mathsf{pp}, \mathsf{c_y})$
**if** $\mathsf{sol} \ge p/2$ **then**
  $\mathsf{sol} \leftarrow \mathsf{sol} - p$
**return** sol

*Figure 12 MCIPFE algorithm*

For the detailed description of the algorithmic pair (Gen, Solve), we refer the reader to the definition of the Decisional Diffie-Hellman with an Easy Subgroup problem as presented in [17]. Moreover, the correctness of the protocol and selection of the deviation parameter σ is analyzed in the context of the security guarantees, also discussed in [17].

This construction being similar to the previous one, the curator starts by running the Setup algorithm to generate the public parameters $(p, \tilde{s}, f, g, h)$, the secret keys $\boldsymbol{s}, \boldsymbol{t}$ and the public keys $h_1, \ldots, h_n$. As mentioned earlier, the user chooses a label ℓ and encrypts data with Encrypt, generating a ciphertext $\boldsymbol{c} = (c_0, c_1, \ldots, c_n)$, where $c_0$ is a hint used for the decryption. Now, consider an analyst wishing to retrieve $\langle \boldsymbol{y}, \boldsymbol{x} \rangle$ for a vector $\boldsymbol{y}$. She sends the curator a request **y** for the ciphertext $\boldsymbol{c}$ of $\boldsymbol{x}$, which provides the functional decryption key $dk_y = (\sum s_i\, y_i\, \ell_i, \sum t_i\, y_i\, \ell_i)$ — output of KeyGen, for the label ℓ under which $\boldsymbol{x}$ is encrypted. By running the Decrypt algorithm, the analyst can finally retrieve the desired quantity $\langle \boldsymbol{y}, \boldsymbol{x} \rangle = y_1\, x_1 + \ldots + y_n\, x_n$.

### 6.4.3 Verifiable Decryption for MCIPFE

We now focus on the verification steps we have added to **Protocol 2**. We propose a method that an analyst can use to verify the correctness of the decryption key $dk_y$ provided by the curator. In case the analyst receives an incorrect value for $dk_y$ during a transaction, he can either cancel the transaction or request a refund from the curator.

***Functional Decryption Key Verification***: A cryptographic protocol needs to meet several security assumptions to guarantee data privacy. Among these requirements, achieving ciphertext indistinguishability guarantees that a ciphertext does not reveal any information

about the plaintext. However, in such cases, how can an external user confirm that a given ciphertext is a valid encryption? This question is particularly important in the field of operations on encrypted data, such as functional encryption or homomorphic encryption, where an external party requests the output of a specific function. This question has been widely studied [12] due to the growing trend of multi-party computation (MPC).

In Figure 13 we propose a verification method for **Protocol 2** based on the discrete logarithm (DLOG). This algorithm is followed by a ZKP on DLOG to guarantee no additional leakage is produced on the messages or the secret key. Given an encryption $c$ of the user's data, assume that an analyst wishes to purchase the encryption $\prod_{\{i=1\}}^{n} c_i^{\{y_i \ell_i\}}$. Recall that this encryption satisfies the equality

$$\prod_{i=1}^{n} c_i^{y_i \ell_i} = f^{\langle \mathbf{y}, \mathbf{x} \rangle} g^{r \sum s_i y_i \ell_i} h^{r \sum t_i y_i \ell_i} = f^{\langle \mathbf{y}, \mathbf{x} \rangle} \cdot \mathrm{g_y}$$

To verify that the analyst retrieves the correct result, it is sufficient to verify that the decryption key $dk_y$ is indeed the right decryption key for the requested vector $y$. We thereunder design the Figure 13, which allows an analyst to check this correctness exclusively using the information known to her.

**Verify** $(\mathsf{mpk}, \mathsf{dk_y}, c_0 = (g^r, h^r), \ell)$:
Encode $(\ell_1, \ldots, \ell_n) \leftarrow \mathsf{Ecd}(\ell)$
Compute $v \leftarrow \prod_{i=1}^{n} h_i^{y_i \ell_i}$
Compute $\mathrm{g_y} \leftarrow \prod (g^r)^{s_y} (h^r)^{t_y}$
**return** $\mathbb{1}\left[\log_g(v) = \log_{g^r}(\mathrm{g_y})\right]$

*Figure 13 MCIPFE verification algorithm.*

To perform the verification, the analyst needs access to the hint $c_0$ of the ciphertext of interest, with both the public and the functional decryption key. These are all public, hence the verification does not impact the security of the protocol. Considering her vector $y$, the analyst

$$\prod_{i=1}^{n} h_i^{y_i \ell_i} = \prod_{i=1}^{n} \left(g^{s_i} h^{t_i}\right)^{y_i \ell_i} = \prod_{i=1}^{n} \left(g^{s_i + \alpha t_i}\right)^{y_i \ell_i} = g^{\sum_{i=1}^{n} y_i \ell_i (s_i + \alpha t_i)}$$

computes:

On the other hand, mark that:

$$\mathrm{g_y} = g^{r s_y} h^{r t_y} = g^{r \sum_{i=1}^{n} s_i y_i \ell_i} h^{r \sum_{i=1}^{n} t_i y_i \ell_i} = g^{r \sum_{i=1}^{n} y_i \ell_i (s_i + \alpha t_i)}$$

The analyst accepts the functional decryption key if $log_{\{g^r\}}(g_y) = log_g(v) for\ v = \prod_{\{i=1\}}^{n} h_i^{\{y_i \ell_i\}}$.

***ZKP on DLOG***: Since the discrete logarithm is not easy in G, the analyst cannot compute $log_{\{g^r\}}(g_y)\ or\ log_g(v)$. However, by using a ZKP she can ensure that the two quantities are indeed equal without leaking information about their actual values.

***Decryption Key Verification for Multi-Users***: Considering the multi-users, we would like to adapt the verification method to guarantee the analyst the correctness of the decryption key. However, the verification process for multi-users is not as straightforward as the single-user construction, and Figure 13 cannot be extended naturally to the multi-user construction. We therefore propose to substitute the method in with a system of n ZKPs. This is not prohibitive as n ciphertexts $c_i$ have to be transmitted anyway, but does constitute a better method for multi-user key verification. Recall that, for $s = (s_1, \dots, s_n)$, $t = (t_1, \dots, t_n)$ hidden to the analyst, $g^{\{r_i\}}$ for $i \in [n]$ included in $c_0$, and $y = (y_1, \dots, y_n)$ generated by the analyst, the decryption key is of the form $dk_y = g^{\{\langle r \circ y, s \rangle\}} h^{\{\langle r \circ y, t \rangle\}}$, where $\langle r \circ y, s \rangle = \sum r_i y_i s_i$ and $\langle r \circ y, t \rangle = \sum r_i y_i t_i$. Following, we claim that it is sufficient for the analyst to know that the correct secret keys $\mathbf{s}, \mathbf{t}$ were indeed used for the generation of $dk_y$ . Hence, we rely on the following ZKP:

$$\pi_i = NIZK\{(s_i, t_i) : h_i = g^{s_i} h^{t_i} \wedge \mathsf{dk_y} = g^{\langle \mathbf{r} \circ \mathbf{y}, \mathbf{s} \rangle} h^{\langle \mathbf{r} \circ \mathbf{y}, \mathbf{t} \rangle}\}, i \in [n].$$

# 7. Decentralized Knowledge Base

The **decentralized Knowledge Base** *(hereinafter KB)* component is designed to store and manage detailed descriptions of resources and their capacities within a swarm-based orchestration ecosystem. The system plays a pivotal role in the Swarmchestrate ecosystem, as it enables efficient matchmaking and dynamic reconfiguration decisions. Main functionality is to maintain comprehensive, up-to-date records of resource attributes that persisted during the ecosystem life span. The KB incorporates Data Models that allow indexing and querying persisted information attributes, leveraging different data store types, ensuring that information for decisions is provided in a timely manner.

The system supports **decentralized information resource management** leveraging decentralized nodes, in the form of agents that are aimed at forming a decentralized network of information. This allows each agent to operate either in a local first or in a decentralized approach contributing to critical decisions. The system interacts closely with core components of the Swarmchestrate, including the Identity & Role Management system *(hereinafter IDM)*, EMS, and Orchestrate System. The key characteristics of information ingestion consider a variety of inputs such as TOSCA-based application descriptions, deployment specifications *(including new requests or state changes)*, resource capacity data, and operational metrics. The KB system acts as a responder of information, thus, according to the action received, for persisting or requesting information, relative output is populated as a response providing for e.g. status updates on swarm resources and capacity availability. This dynamic exchange of information is deemed necessary for the ecosystem and allows the orchestration system to prescribe necessary adaptations on runtime changes, maintaining optimal performance and efficient utilization across the decentralized environment.

## 7.1 Background and related work

**Decentralized Data Store Frameworks** provide mechanisms to store, query, and manage data without reliance on a central authority. *OrbitDB* trust [30], is a multifold peer-to-peer database that utilizes IPFS for storage and CRDTs for conflict-free data replication. *BigchainDB* [31], offers a combination of blockchain characteristics, with decentralized control and immutability. *CovenantSQL* [32] is an SQL like store, built on blockchain technology, ensuring data integrity and tamper resistance. *GunDB* leverages Graph-based data models to offer real-time read-writes designed for peer-to-peer *(P2P)* interaction [33]. With regards to **Storage frameworks like** *IPFS,* a peer-to-peer hypermedia protocol allows to host and share content in a decentralized manner. **Decentralized Communication frameworks** like *libp2p* provide a modular stack using a set of protocols for building communication, transportation and discoverability across decentralized systems. The incorporation of gossip-based protocols [16], enhances message dissemination, improving scalability and robustness. According to research contributions and respective demands for Swarmchestrate, our characterization of gaps and the addressability of them focuses *on Scalability, Authentication & Authorization, Privacy & Confidentiality, Integration & Compatibility, Performance & Efficiency and Collaboration* which shall be examined and implemented to surpass limitations of traditional centralized data stores and support the proposition of a lightweighted decentralized KB, which incorporates efficient decentralized data storage layer based upon IPFS and libp2p framework, so that KB agents and respective components in the foreseen decentralized environment can communicate. Integration and interoperability as considered by [34] examines the issue of integrating decentralized systems with traditional databases, focusing on the necessity of middleware APIs.

The KB addresses several key limitations inherent in traditional centralized data systems. To overcome scalability bottlenecks, the KB incorporates agents, enabling localized orchestration and horizontal growth without overloading a central node. We use CRDT-based replication and peer-to-peer synchronization to eliminate single points of failure, ensuring that orchestration can continue even during partial outages. For authentication and authorization, we integrate with the respective IDM system, using identity mechanisms such as DIDs and verifiable credentials. Amongst agents of the KB, privacy and confidentiality are preserved through selective disclosure and attribute-level access control, minimizing the exposure of sensitive metadata. Integration and compatibility challenges are resolved by exposing middleware APIs, allowing seamless interaction with the orchestration system. The respective layer provides a transparent layer of accessible information regardless of the location that the data relies within the KB subcomponents. Finally, the system supports dynamic collaboration and data sharing among KB agents, enabling resilient and brokerless communication across the decentralized orchestration environment.

## 7.2 Definition of roles in Swarmchestrate

The roles that are closely related to the operation and interaction of the Decentralized KB, consider interaction under an Ingress and Egress Integration framework.

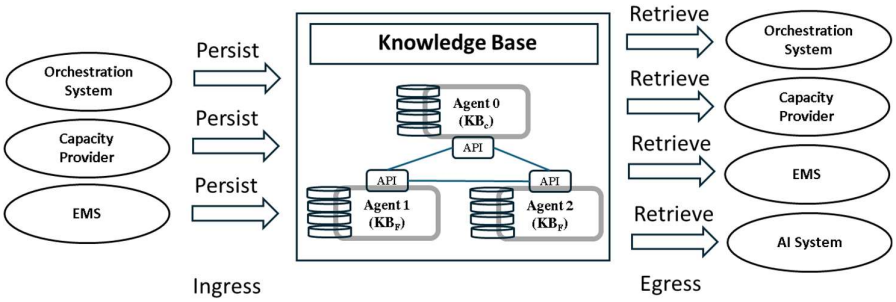



*Figure 14 Logical representation of components interacting with KB*

The ingress and egress middleware touchpoints are designed to handle RESTful payloads for ingesting structured orchestration metadata into the KB. These payloads encapsulate application descriptions, deployment plans and capacity data.

The Orchestration system is responsible for defining and executing application deployment plans based on high-level service descriptors that are based upon TOSCA templates in yaml. The aim is to provide a bidirectional flow leveraging the information of the KB, that continuously adapt to deployment strategies in alignment with current swarm conditions and policies defined in the orchestration logic. Therefore, with regards to the interaction with the KB, the Orchestration system, acts both as a producer and consumer of information. The anticipated scenarios consider pushing application descriptions, deployment configurations, and orchestrator-specific templates (i.e. ingress), whilst also retrieving processed deployment plans, resolved resource mappings, and status updates (i.e. under egress).

The contribution of the Capacity Provider is to supply real-time infrastructure and resource information to the KB, such as dynamic attributes such as CPU availability, memory size, network proximity, resource status etc, contributing detailed capacity descriptions under a TOSCA based format.

The Event Monitoring System (hereinafter EMS) ingests operational telemetry, SLO violations, and service-level aggregated events and persist them in the KB, logging resources behaviour. These event-based information addresses the context knowledge for runtime intelligence, enabling proactive reconfiguration, fault detection, and SLO monitoring. From the egress side, the EMS responds to the KB for the successful retrieval of root-cause indicators, or metrics correlated with performance thresholds.

In principle each component, Orchestration Subsystem, Capacity Provider, and EMS interfaces with the KB to either store or retrieve critical artifacts. Files such as full application descriptors or orchestration templates are base64-encoded and embedded in JSON, ensuring compatibility and integrity during transfers. To that end KB incorporates different types of Data Stores that serve a specific information retrieval scenario and need within the Swarmchestrate.

The following tables outline the Egress and Ingress activities, as depicted also in Figure 14 compared to the roles of systems incorporated in Swarmchestrate:

*Table 1 Ingress Integration description*

| Context | Datastore | Purpose |
|---|---|---|
| Orchestration system | App Descriptions (ADT) Datastore | Store full application description based on TOSCA YAML |
| Orchestration system | Imported OpenTofu/TOSCA Templates Datastore<br>-Swarm status Information- | Raw or partially processed orchestrator templates |
| Capacity Provider | Capacity Descriptions Datastore | Capacity templates describing available resources in TOSCA |
| Orchestration system | Deployment/Release Plans Datastore<br>-capacity status information- | • Store prepared deployment plans parsed by TOSCA<br>• Retrieving the status of existing capacities<br>• Or changing the state of the capacities for release<br>• Available resources |
| EMS | Event / Violation History Datastore | Capture composite SLO violations and events (Metadata) |

*Table 2 Ingress Integration specifics*

| Context | Datastore | Purpose |
|---|---|---|
| Orchestrator system | Capacity Descriptions Datastore | Receive file with Capacity templates describing available resources in TOSCA |
| Orchestrator system | OpenTofu/TOSCA Templates Datastore | Receive application deployment info using OpenTofu, i.e. document id |

| Context | Datastore | Purpose |
|---|---|---|
| **Orchestrator system** | **Deployment/Release Plans Datastore** | Receive available resources for Deployment |
| **Orchestrator system** | **App Descriptions (ADT) Datastore** | Store full application description based on TOSCA YAML |
| **AI system** | **Event / Violation History Datastore** | Respond providing composite SLO violations and events (Metadata) |

## 7.3 Initial System Design overview

We propose a **paradigm of a decentralized ecosystem** incorporated with various entities, built under *the concept of decentralized agents that considers a reliable, secure, and transparent knowledge management infrastructure* [35]. We consider the operation of the data store to be enabled under a **peer-to-peer (P2P) architecture**. We have used Kubernetes[2] clusters under different Cloud tenants, supporting high availability for the prototype container deployment but also for the unification of operation under a swarm. For the edge perspective we use a fork of k8s, k3s[3] that is interoperating with k8s cluster and assures edge-to-core uniformity of access. The Kubernetes control plane oversees the swarm's infrastructure, coordinating the deployment of several components across different tenants.

The KB considers resource details such as type, location, status, and capacity; workflow and orchestration metadata for scheduling and execution; relationships between resources of the swarm, workflows, networks, and storage; monitoring and log data for system operations and error severity; networking configurations like CIDR blocks and regions; and temporal data such as timestamps for creation, updates, allocations, and deallocations to maintain operational timelines. The KB components are deployed in appropriate proximity as depicted within each swarm to effectively serve devices or applications. It acts as a responder when information is requested or when persisting resource metrics and querying them is essential. To calculate proximity and deploy each KB agent accordingly, we consider leveraging information from K8s

[2] Kubernetes. (n.d.). *An open-source system for automating deployment, scaling, and management of containerized applications*. Retrieved December 20, 2024, from https://kubernetes.io.
[3] K3s. (n.d.). *A lightweight Kubernetes distribution for developer workstations, IoT, and edge computing*. Retrieved December 24, 2024, from https://k3s.io/.

APIs, *i.e. metric store for existing resource information of deployed containers*, which are evaluated based on a heuristic tournament selection match between candidate points of the swarm. The swarm incorporates different Agents particularly reflected by the name convention of KB as can be seen in the above figure. These KB agents are interconnected under a common middleware layer and P2P topic of *"KnowledgeBase"* and guarantee the availability of information towards other resources, at the requisite time and location.

The Decentralized KB system is supported by a logical and physical component layered structure. The logical structure is translated to a description of decentralized, loosely coupled agents and their respective services *(physical layer structure)* that operate in tandem to collect, store, analyse, and share knowledge.

## 7.4 System Design Logical layer

Considering **the logical layer**, these agents have different roles and are clearly separated to support internal mechanisms and address decentralized challenges in the ecosystem. There are two different types of agents, characterized by their operational nature, referred as **Coordinator ($KB_c$)** and **Follower Agents ($KB_f$)** considering a self-explanatory meaning for leadership election assuring KB consistency. These agents are established or elected at runtime or during the KB lifetime to ensure scalability, fault-tolerance, and policy-driven orchestration. These agents act as decentralized computation and decision-making units with defined responsibilities across the data lifecycle. The $KB_c$ are establishing the trajectory for the $KB_f$, who adhere to their directives. At the beginning $KB_c$ is elected by utilizing a flag as Coordinator to ascertain their function within the network. Among others, the agent designated with this flag is tasked with initializing and overseeing the overall *"KnowledgeBase"* interactions. Agents identify and establish connections in an automated manner, with the coordinator via multicast DNS[4] or bootstrap peers according to the deployed scenario. KB agents share resource metadata during initial discovery and select a coordinator depending on criteria, like uptime, latency, or resource availability. In the case of a coordinator agent loss, an automated re-election of a new coordinator is performed, ensuring uninterrupted continuity.

[4] Multicast DNS. (n.d.), Referred as mDNS. *Wikipedia*. Retrieved December 20, 2024, from https://en.wikipedia.org/wiki/Multicast_DNS.

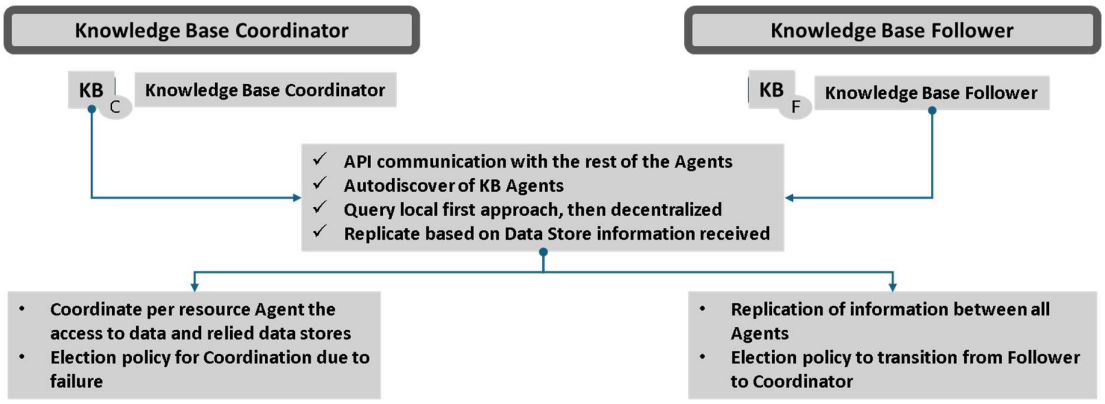



*Figure 15 High level overview of the Coordinator and Followers Agents under the decentralized KB component*

The Coordinator Agent primary function is to aggregate, reason, and orchestrate knowledge flows across multiple Follower Agents. Key responsibilities incorporate the following functionalities:

- Query Aggregation and Ranking: Collects respective responses from Follower Agents aggregates or deduplicates in case of decentralized queries execution. Ranks respective information received from the decentralized network of KB agents, considering operational health based on semantic weights, relevance, or policy rules, persisting in a shared table across all agents – for Coordinator election in case of failure.
- Global Synchronization Control: Manages periodic consistency checks and orchestrates peer-to-peer synchronization schedules among Follower Agents, ensuring knowledge convergence.
- Topology-Aware Decision-Making: Maintains awareness of the network and resource graph to make topology-optimized recommendations.
- Leader Election & Failover: Participates in and often coordinates election protocols to designate fallback Coordinators in case of failure.

Follower Agents are collocated with a Resource Agent or runs near it *(logically or physically)* and is tasked with:

- Local first or decentralized Query Execution: Responds to match requests locally by evaluating conditions against the KB agent's capabilities. In case information has not persisted in the respective agent's data store a decentralized query process is attempted over the decentralized network of agents.
- Metadata Store Management: Maintains and updates the persistent or ephemeral metadata store.
- Selective Data Store Sync: Pushes updates to peer Follower Agents or upstream Coordinators based on relevance, event triggers, or query demand. The aim is merely to replicate information i.e. metadata and/or other data with respect to internal mechanisms such as information on peers of the network and metrics that may affect the Leader-Election process.

- **Failure Isolation:** Operates independently in failure-prone or partitioned environments, ensuring localized continuity of decision-making even if disconnected from the coordinator.

## 7.5 Fault Taulerance

To promote fault tolerance, we utilize Libp2p's Pub/Sub for Agent coordinator election. KB agents share resource metadata during initial discovery and select a coordinator depending on criteria, like uptime, latency, or resource availability. In the case of a coordinator agent loss, an automated re-election of a new coordinator is performed, ensuring uninterrupted continuity.

In Figure 16 we demonstrate the process for a Coordinator election in the KB system architecture as part of a decentralized network. In this example, the system considers a Coordinator agent ($KB_C$), three Follower agents *($KBF_1$, $KBF_2$, $KBF_3$)* under a topic for the election process. The process begins with the KBc broadcasting a reputation score calculation activity to the follower agents.

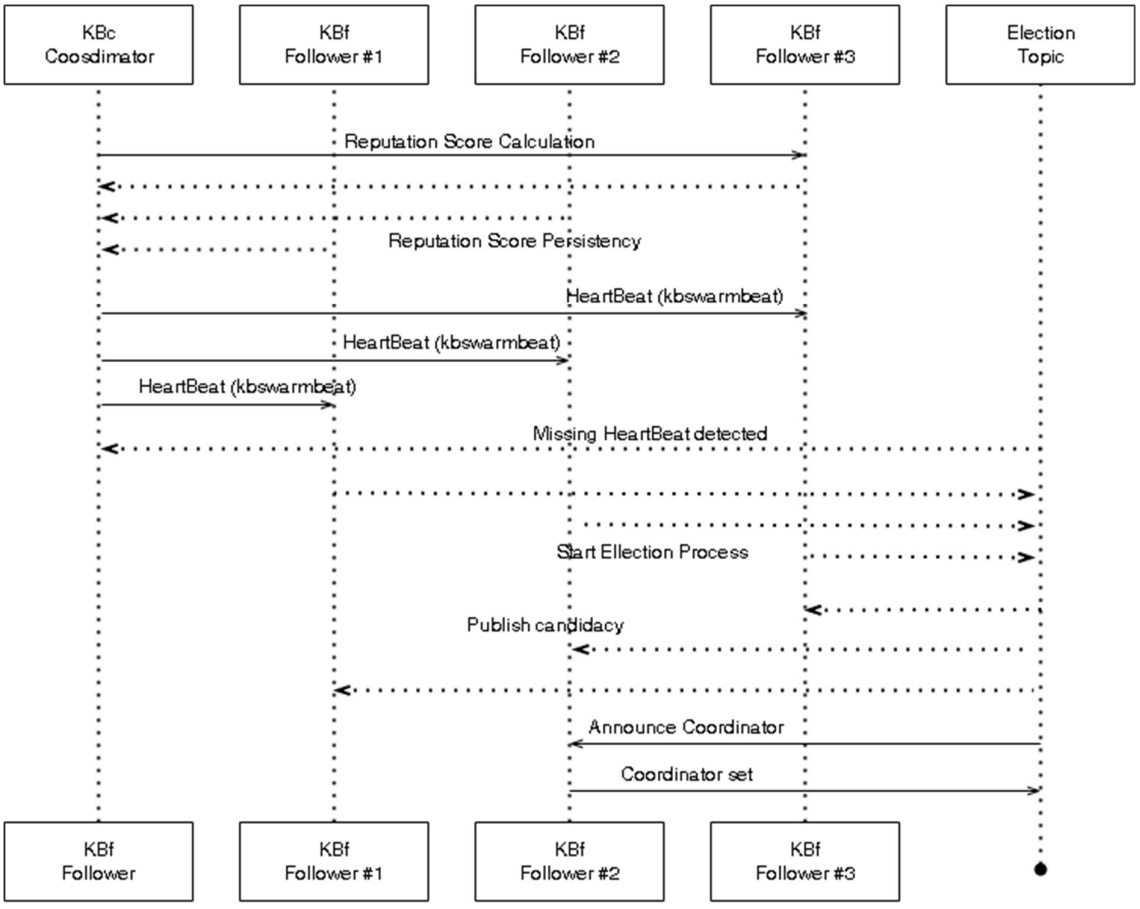



*Figure 16 Election process for KB Coordinator*

The aim is for each of the agents to calculate a score *(hereinafter stability reputation)* that reflects various operational metrics, such as uptime, system resource availability, networking discrepancy, and historical performance. An example of the Reputation score calculation with specific weights follows:

$$Agent_{Reputationscore} = (0.10 \times \text{Uptime}) + (0.15 \times \text{LeadershipCount}) + (0.25 \times \frac{1}{\text{Latency}}) + (0.10 \times \text{AvailableMemory}) + (0.10 \times \text{Responsiveness}) + (0.05 \times (1 - \text{CPULoad})) + (0.05 \times \text{StorageAvailability}) + (0.20 \times \text{ProjectedHealthScore})$$

The reputation score calculation is a repetitive internal KB agent process, yet transparent to the rest of the Swarmchestrate ecosystem. It is configured to run once per a five-minute interval to assure proactive population of information for potential coordinator election in case of a failure is faced in the current KB Coordinator agent. This process considers the following factors to be recalculated under this interval basis.

- Uptime (10%): Measures the duration of the agent remained continuously online without failures. A higher value implies better stability and reliability.
- Leadership Count (15%): Indicates the number of times the agent has successfully served as a leader in past election cycles. This shows leadership experience and trustworthiness.
- Latency (25%): is a dominant factor in distributed coordination. Leaders with low latency improve consensus round efficiency, replication time, and system availability. Represents the physical or logical proximity of the agent to target agents, i.e. agents ping peers or measure round-trip times (RTT) and use the results to infer relative closeness.
- Available Memory (10%): Quantifies the currently available system memory on the agent. Higher memory availability suggests the agent is well-resourced and capable of handling coordination.
- Responsiveness (10%): Measures how quickly the agent responds to events such as heartbeat messages, or queries.
- CPU Load (5%): Reflects the agent's current versus historical processing burden. Lower CPU load is preferred, as it indicates available compute capacity for handling leadership responsibilities.
- Storage Availability (5%): Measures the free disk space on the agent. Sufficient storage is important for agents managing coordinator operations, such as replication.
- Projected Health Score (20%): This score estimates the future performance of an agent by analysing the trend of key operational metrics over the last day repetition, using linear regression. It is designed to proactively assess whether an agent's health is improving, stable, or deteriorating. The respective score is calculated as follows:

$$\text{ProjectedHealthScore} = \frac{1}{n}\sum_{i=1}^{n} \text{normalize}\left(-\frac{dM_i}{dt}\right) \text{where:}$$

$$M_i \in \{\text{Latency}, \text{CPULoad}, \text{Responsiveness}, \text{AvailableMemory}\}$$

$$\text{normalize}(x) \in [0,1] \text{ scales the input trend to a bounded score}$$

$$-\frac{dM_i}{dt} \text{ is the slope (rate of change) over time, inverted if high values are undesirable}$$

**ProjectedHealthScore** falls within the *range [0.0, 1.0]*, where candidates between *[0.8-1.0]*, are high confidence stable agents, whereas candidates 0.35-0.45 suggesting a potential no fit for candidate. Respective weights rationale and effective interval of recalculation has been based on the directions articulated under similar election mechanisms, of *DAG-BFT* [36]*, PrestigeBFT* [37] *and FRLLE* [38].

Each follower persistently stores the reputation score data and publishes across the topic *(candidacy)*, for later use in decision-making. Every agent maintains the same table of reputation amongst all agents at ensuring consistency across the decentralized KB. Subsequently, the coordinator sends regular heartbeat messages under the same topic, to all followers, which act as a liveness check. These heartbeats are individually acknowledged by each follower. If one or more followers do not receive a heartbeat within the expected time window, they infer that the coordinator may be unavailable or has failed. Upon detecting the coordinator's failure, the follower agents initiate the election process. Each follower according to the exchanged candidacy and based on the reputation score proposes the elected candidate. The agent with the highest reputation is elected as the new coordinator. This final step completes the election cycle, ensuring system continuity without centralized failure.

## 7.6 System Design Physical layer

Considering the physical layer of each KB agent, regardless of the nature of operation, the following figure depicts the internal components of the Agent. Each agent encapsulates several functional layers, each with a specific role, ranging from API exposure to peer-to-peer networking. Starting from the **Middleware layer**, standardized interfaces are exposed to external systems interacting with the KB Agent using various APIs. The **IPFS API** Interacts with the IPFS stack amongst agents for data retrieval, CID resolution, and DAG manipulation as well as information to be leveraged for internal mechanisms using the ipfs pub/sub protocol. The **Shell API** is a user-based access tool to support command-line or container-internal administrative tasks for debug only reasons. The **REST API** incorporates the integration points for the KB agent to communicate with the rest of the ecosystems and follower queries *(e.g., /knowledge/query, /metadata/update)*. The API supports payloads considering different type of query languages for document stores, event stores, key-value stores but also ANSI SQL based stores.

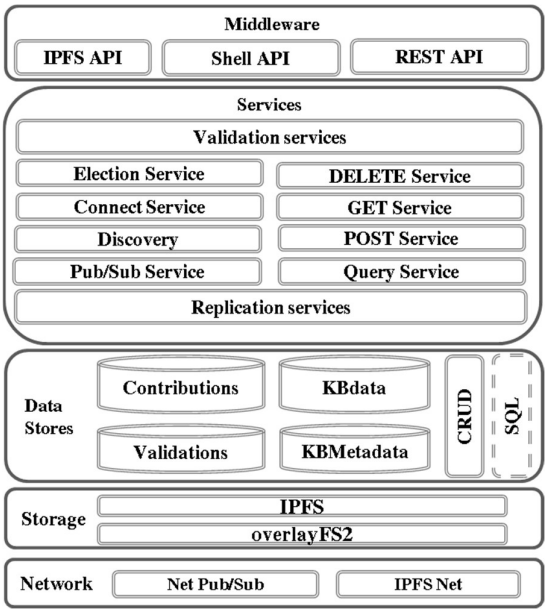



*Figure 17 Internal components of the physical structure of the KB*

The **Service layer** considers a number of **services for validation** of data received and persisted as well as **replication services** for Data store synchronization. The most critical validation services for ensuring consistency, integrity, and proper processing of incoming data and

requests are the **Election Service** *(supports leader election in decentralized setups of agents)*, **Connect and Discovery Services** *(used for establishing peer topology and initiate syncs via libp2p)*, **Query Services** *(supporting CRUD-style or ansi-SQL like operations)* **Pub/Sub Service** *(Interfaces with EMS or native libp2p topics for event-driven updates)*. The replication services, handle information propagation between KB Agents which are supported by hash-driven sync mechanisms, partial or full DAG[5] syncs, deduplication and traceability *(using IPFS CIDs).*

The **Data Store layer** supports decentralized and structured data management incorporating various Data Stores for information persistence. This layer incorporates data repos of Contributions *(event logs)*, Validations *(data integrity checks)*, KBdata *(resource metrics knowledge)*, and KBMetadata *(metadata of the KBdata)*, supported by CRUD and SQL interfaces for structured access. While Data Stores in offer **data management off-chain**, the **"Contributions" event Log** is designed as an **immutable append-only store** to preserve specific data. Respective data consider metadata from KB transactions and events from verifiable credentials as occurred and persisted to the KB from the IDM system. The key purpose is to provide tampered-free protection of specific data, under **a private blockchain based environment** guaranteeing traceability and auditability, in an on-chain manner. All data stores support CRUD operations across both decentralized and structured data backends. IPFS is utilized to provide content-addressed storage for traceable entries, ensuring distributed consistency and replication. At the same time, SQL-based layer is used to enable efficient indexing, metadata retrieval, and support for relational operations, such as joins for scenarios that are required for advanced query execution and orchestration alignment. We incorporate, different data storage models as **document stores**, **key-value stores**, and **event logs,** but also developed an SQL parser based by extending the custom implementation of OrbitDB and SQLite in GO. Document stores allow for flexible schema design, while key-value stores contribute to high performance for lookups. To address RDBMS query support, we have considered an SQL GO wrapper to abstract the underlying storage model. Respective layer incorporates declarative language that extends SQL, providing different kinds of tables and views at the schema level, along with sensitive columns and their minimum granularity level of aggregations. Other interactions can be achieved using CRUD and SQL like commands to receive information from KBdata and KBMetadata datastores. The information can be retrieved using respective http APIs supporting either an SQL statement in the POST payload or any CRUD command. These mechanisms guarantee that all agents and entities in the swarm contribute to generate a unified Data Store supporting the reliability of the Knowledge on this case within various locations of the swarm.

The **Storage layer** at its core, uses IPFS as the decentralized file system, persisting entries as Directed Acyclic Graphs (DAGs) that are content-addressable *(ability to reproduce data states across agents)*, allowing traceability via unique CIDs. We leverage IPFS frameworks written in GO, to serve as the backbone for data that is physically moved or stored in a noncentralized

[5] Merkle Directed Acyclic Graphs (DAGs), https://docs.ipfs.tech/concepts/merkle-dag/

location. In parallel, overlayFS2 is employed to support a layered file system within the container runtime.

The **Network layer** supports the agent's peer communication stack, merely using Pub/Sub protocol for lightweight topic-based propagation, and IPFS networking for a direct P2P message routing, DHT lookups, KB agents peer discovery.

# 8. Swarmchestrate Knowledge Management Layer

The Knowledge-Management layer in Swarmchestrate (see Figure 18) is modelled as four tightly-coupled but clearly-bounded systems. Together they ingest information coming from the rest of the platform, turn it into ranked, trusted, and verifiably-attested knowledge, and then expose that knowledge back to other layers.

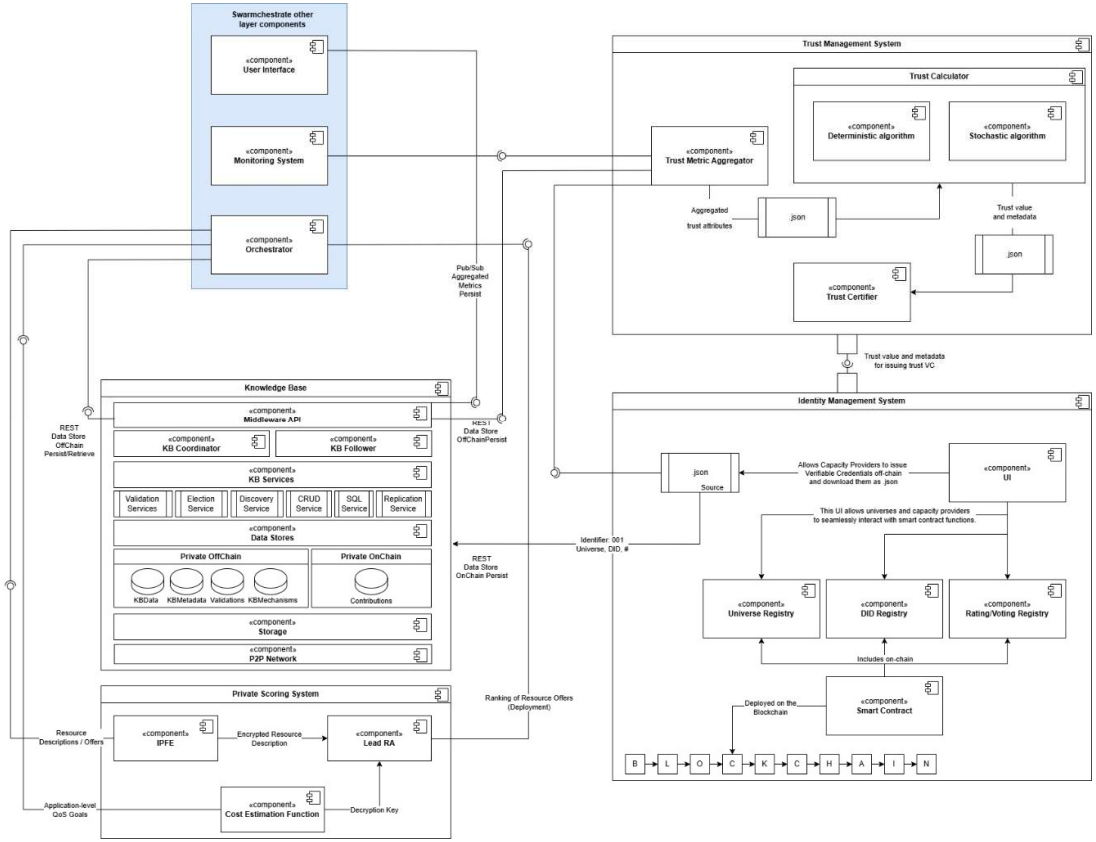



*Figure 18 Swarmchestrate Knowledge Management Layer*

- **Knowledge Base:** serves as the knowledge core of the Swarmchestrate platform. It allows data to be persisted in a dual-store architecture: (i) off-chain document and metadata repositories for low-latency retrieval, and (ii) an on-chain contributions ledger (IPFS-backed) that guarantees immutability and provenance. In the Knowledge Management Layer, it mainly communicates with the Trust Management System and the Identity Management System.
- **Trust-Management System**: ingests the KB's aggregated attributes and subjects them to a trust-calculation pipeline comprising deterministic rule-based evaluation or stochastic/probabilistic modelling. Results are passed to a Trust Certifier, which encapsulates each score together with its supporting evidence in a digitally-signed verifiable credential. This cryptographic attestation renders the trust metric tamper-evident and auditable, thus enabling rigorous differentiation between trustworthy and non-trustworthy resource providers or capacities.
- **Identity-Management System:** binds identities to DIDs and anchors them on a permissioned blockchain. Through a lightweight UI, capacity providers retrieve, issue, and manage VCs. Moreover this system, complements the Trust Management System in the process of issuing verifiable credentials for trust. The result is an end-to-end, cryptographically-verifiable identity and reputation substrate that enhances transparency, accountability, and traceability across the multi-stakeholder ecosystem.
- **Private Scoring System:** facilitates privacy-preserving selection of deployment targets. Providers submit encrypted resource offers processed via an Inner-Product Functional Encryption (IPFE) engine; a Lead Resource Agent obtains decryption keys solely for the relevant scalar products that encode application-level QoS goals. The subsequent Cost-Estimation Function generates a ranked list of candidate capacities without ever disclosing sensitive commercial data. The orchestrator thus receives an optimised, privacy-compliant ordering, and the resulting deployment decisions—as well as post-deployment metrics—are reinjected into the KB, completing a closed, evidence-based feedback loop.

# 9. Conclusion

Deliverable D3.1 establishes both the conceptual foundations and a concrete architectural roadmap for trustworthy, secure, auditable, and privacy-respecting orchestration within Swarmchestrate. By formalising a multidimensional trust model and designing four complementary components (i.e. Trust Management System, Identity and Role Management System, Private Scoring, and the Decentralised Knowledge Base) the Knowledge Management Layer now possesses a set of coherent mechanisms that enable trustworthy and secure interactions in the system as well as decentralized and privacy preserving knowledge management and orchestration. Preliminary prototypes confirm the technical feasibility of the approach and highlight key integration points for future work.

# 10. Bibliography

[1] K. Tamas, U. Amjad, K. Jozsef, K. Dimitris, T. Alexandros, V. Yiannis, A. Jörn and K. Vogel, "D2.1 Decentralised Orchestrator Early Release," 2025.

[2] J. B. Rotter, "Generalized expectancies for interpersonal trust," *American psychologist,* 1971.

[3] D. Gambetta, "Trust: Making and breaking cooperative relations," *Basic Blackwell,* 1988.

[4] J. S. Coleman, Foundations of social theory, 1990.

[5] J. B. Barney and M. H. Hansen , "Trustworthiness as a source of competitive advantage," *Strategic management journal,* vol. 15, pp. 175--190, 1994.

[6] R. Holton, "Deciding to trust, coming to believe," *Australasian journal of philosophy,* vol. 72, no. 1, pp. 63-76, 1994.

[7] S. P. Marsh, *Formalising trust as a computational concept,* Stirling: University of Stirling, 1994.

[8] R. C. a. D. J. H. a. S. F. D. Mayer, "An integrative model of organizational trust," *Academy of management review,* vol. 20, no. 3, pp. 709-734, 1995.

[9] R. J. Lewicki, "Trust in relationships: A model of development and decline," *Conflict, Cooperation and Justice: Essays Inspired by the Work of Moreton Deutsch/Jossey-Bass,* 1995.

[10] R. a. D. T. M. a. P. M. M. Bhattacharya, "A formal model of trust based on outcomes," *Academy of management review,* vol. 23, no. 3, pp. 459-472, 1998.

[11] T. Grandison and M. Sloman, "A survey of trust in internet applications," *IEEE Communications Surveys & Tutorials,* vol. 3, no. 4, pp. 2-16, 2000.

[12] D. H. McKnight and N. L. Chervany, "What is trust? A conceptual analysis and an interdisciplinary model," in *AMCIS 2000 Proceedings. 382*, 2000.

[13] B. Esfandiari and S. Chandrasekharan, "On how agents make friends: Mechanisms for trust acquisition," in *Proceedings of the fourth workshop on deception, fraud and trust in agent societies*, Montreal, 2001.

[14] T. W. Simpson, "What is trust?," *Pacific Philosophical Quarterly,* vol. 93, no. 4, pp. 550-569, 2012.

[15] L. Atzori, A. Iera and G. Morabito, *Social internet of things: turning smart objects into social objects to boost the IoT,* IEEE Internet of Things Newsletter, 2014.

[16] W. Sherchan, S. Nepal, J. Hunklinger and A. Bouguettaya, “A trust ontology for semantic services,” in *2010 IEEE International Conference on Services Computing*, 2010.

[17] C. Nuoskala, R. Rabbaninejad, T. Dimitriou and A. Michalas, “Fe [r] chain: Enforcing fairness in blockchain data exchanges through verifiable functional encryption,” in *Proceedings of the 29th ACM Symposium on Access Control Models and Technologies*, 2024.

[18] D. Boneh, A. Sahai and B. Waters, “Functional encryption: Definitions and challenges,” in *Theory of Cryptography: 8th Theory of Cryptography Conference*, 2011.

[19] B. Waters, “A punctured programming approach to adaptively secure functional encryption,” in *Annual Cryptology Conference*, 2015.

[20] S. Goldwasser, Y. T. Kalai, R. A. Popa, V. Vaikuntanathan and N. Zeldovich, “How to run turing machines on encrypted data,” in *Advances in Cryptology--CRYPTO 2013: 33rd Annual Cryptology Conference*, 2013.

[21] S. Goldwasser, S. D. Gordon, V. Goyal, A. Jain, J. Katz, F.-H. Liu, A. a. S. E. Sahai and H.-S. Zhou, “Multi-input functional encryption,” in *Advances in Cryptology--EUROCRYPT 2014*, 2014.

[22] M. Abdalla, F. Bourse, A. De Caro and D. Pointcheval, “Simple functional encryption schemes for inner products,” in *IACR International Workshop on Public Key Cryptography*, 2015.

[23] S. Agrawal, B. Libert and D. Stehle, “Fully secure functional encryption for inner products, from standard assumptions,” in *Annual International Cryptology Conference*, 2016.

[24] M. Abdalla, R. Gay, M. Raykova and H. Wee, “Multi-input inner-product functional encryption from pairings,” in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, 2017.

[25] S. Badrinarayanan, V. Goyal, A. Jain and A. Sahai, “Verifiable functional encryption,” in *International Conference on the Theory and Application of Cryptology and Information Security*, 2016.

[26] D. D. Nguyen, D. H. Phan and D. Pointcheval, “Verifiable decentralized multi-client functional encryption for inner product,” in *International Conference on the Theory and Application of Cryptology and Information Security*, 2023.

[27] J. Chotard, E. Dufour Sans, R. Gay, D. H. Phan and D. Pointcheval, “Decentralized multi-client functional encryption for inner product,” in *Advances in Cryptology--ASIACRYPT 2018: 24th International Conference on the Theory and Application of Cryptology and Information Security*, 2018.

[28] G. Castagnos, F. Laguillaumie and I. Tucker, “Practical fully secure unrestricted inner product functional encryption modulo p,” in *International Conference on the Theory and Application of Cryptology and Information Security*, 2018.

[29] M. Abdalla, F. Benhamouda and R. Gay, “From single-input to multi-client inner-product functional encryption,” in *International Conference on the Theory and Application of Cryptology and Information Security*, 2019.

[30] M. Stonebraker, P. Brown, A. Poliakov and S. Raman, “The architecture of SciDB,” in *Scientific and Statistical Database Management: 23rd International Conference, SSDBM 2011*, Portland, 2011.

[31] M. Kumar, R. Hritu, C. Nisha and S. G. Sukhpal, “Blockchain inspired secure and reliable data exchange architecture for cyber-physical healthcare system 4.0,” *Internet of Things and Cyber-Physical Systems,* vol. 3, pp. 309-322, 2023.

[32] C. Labs, “CovenantSQL: the SQL Database on Blockchain,” Medium, 4 09 2018. [Online]. Available: https://blog.goodaudience.com/covenantsql-the-sql-database-on-blockchain-db027aaf1e0e. [Accessed 14 05 2025].

[33] K. D. C. Lin, P. Maziyar, Y. Tsaiching, T. Kazuhiro and M. Masaru, “Mobile-based traceability system for sustainable food supply networks,” *Nature Food,* vol. 1, no. 11, pp. 673-679, 2020.

[34] R. Venkatesh, D. Alexander, G. Francisco, A. Oliver, B. Divya, O. Lisa, M. Shivram and J. Lav, “Platform Extension Framework (PXF): Enabling Paral-lel Query Processing Over Heterogeneous Data Sources In Greenplum,” in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2020.

[35] G. Georgakakos and V. Yiannis, “A Novel Data Store Supporting the Decentralized Data Management in the Cloud Computing Continuum,” in *International Conference on Advanced Information Networking and Applications*, 2025.

[36] A. Spiegelman, B. Arun, R. Gelashvili and Z. Li, “Shoal: Improving dag-bft latency and robustness,” in *International Conference on Financial Cryptography and Data Security*, 2024.

[37] G. Zhang, P. Fei, T. Sofia and J. Hans-Arno, “Prestigebft: Revolutionizing view changes in bft consensus algorithms with reputation mechanisms,” in *2024 IEEE 40th International Conference on Data Engineering (ICDE)*, 2024.

[38] A. Biswas, A. K. Maurya, A. K. Tripathi and S. Aknine, “Frlle: a failure rate and load-based leader election algorithm for a bidirectional ring in distributed systems,” *The Journal of Supercomputing,* vol. 77, pp. 751-779, 2021.

[39] R. Kripalani, “Demystifying libp2p gossipsub: a scalable and extensible p2p gossip protocol,” 2019. [Online]. Available:

https://archive.devcon.org/archive/watch/5/demystifying-libp2p-gossipsub-a-scalable-and-extensible-p2p-gossip-protocol/.

[40] S. Alexander, A. Balaji, G. Rati and L. Zekun, “Shoal: Improving DAG-BFT Latency And Robustness,” 7 7 2023. [Online]. Available: https://arxiv.org/abs/2306.03058.

[41] J. Chotard, E. Dufour-Sans, R. Gay, D. H. Phan and D. Pointcheval, “Dynamic decentralized functional encryption,” in *Annual International Cryptology Conference*, 2020.

swarmchestrate

# Annex – Trust Taxonomy