-

TRUSTWORTHY, COGNITIVE AND AI-DRIVEN COLLABORATIVE ASSOCIATIONS OF IOT DEVICES AND EDGE RESOURCES FOR DATA PROCESSING

Grant Agreement no. 101136024

Deliverable D3.1
Security, trust and data management for distributed data processing

Programme:	HORIZON-CL4-2023-DATA-01-04
Project number:	101136024
Project acronym:	EMPYREAN
Start/End date:	01/02/2024 – 31/01/2027

Deliverable type:	Report
Related WP:	WP3
Responsible Editor:	CC
Due date:	30/04/2025
Actual submission date:	30/04/2025

Dissemination level:	Public
Revision:	FINAL

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101136024

Revision History

Date	Editor	Status	Version	Changes
20.02.25	Eduardo Cánovas Martínez (UMU) and Marton Sipos (CC)	Draft	0.1	Initial description of Privacy and Security Manager, component description structure
13.03.25	Eduardo Cánovas Martínez (UMU)	Draft	0.2	Extended description of Privacy and Security Manager, DP-ABC Module
19.03.25	Marton Sipos (CC)	Draft	0.3	Initial description of the Edge Storage Service
31.03.25	Ivan Paez (ZSCALE)	Draft	0.4	Initial description of the Decentralized Distributed Data Manager
02.04.25	Marton Sipos (CC)	Draft	0.5	Extended description of the Edge Storage Service, Introduction
08.04.25	Aristotelis Kretsis (ICCS), Ivan Paez (ZSCALE), Marton Sipos (CC)	Draft	0.6	EMPYREAN Architecture, extended description of the Decentralized Distributed Data Manager
14.04.25	Anastassios Nanos (NUBIS), Marton Sipos (CC)	Draft	0.7	Final contributions to component descriptions, preparations for internal review
30.04.25	Marton Sipos (CC) and Aristotelis Kretsis (ICCS)	Final	1.0	Updates based on internal reviews

Author List

Organization	Author
CC	Marton Sipos
ICCS	Aristotelis Kretsis
NUBIS	Anastassios Nanos
UMU	Eduardo Cánovas Martínez
ZSCALE	Ivan Paez

Internal Reviewers

RYAX: Yiannis Georgiou

NUBIS: Christos Panagiotou

Abstract: This deliverable presents the technical outcomes of Task 3.1: “Distributed Trust Management, Trust Propagation and Verification” and Task 3.2: “Data Management for Distributed Data Processing”, between M4 and M15. First, the general EMPYREAN architecture is described, highlighting the role of each of the components developed in the two tasks. This is followed by a detailed technical description of each component, including internal architecture, key workflows, public APIs and planned integrations with other platform components. The relationship of each component with project objectives and KPIs is described as well as the connections to the project’s use cases. Finally, the conclusion describes the work in the context of the project, highlighting future steps.

Keywords: EMPYREAN Architecture, Privacy and Security Manager, DP-ABC Module, Edge Storage Service, Decentralized and Distributed Data Manager, KPIs, Objectives, Workflows, Integration, Cloud-native IoT Device Management

Table of Contents

List of Figures

List of Tables

Abbreviations

AI	Artificial Intelligence
AMQP	Advanced Message Queuing Protocol
APC	Attribute-Based Credentials
API	Application Programming Interface
ATR	Autonomous Towing Robots
AWS	Amazon Web Services
CLI	Command Line Interface
CRI	Container Runtime Interface
CRUD	Create, Read, Update, Delete
CSG	Cloud Storage Gateway
CTA	Cyber Threat Alliance
CTI	Cyber Threat Intelligence
CV	Computer Vision
CVEs	Common Vulnerabilities and Exposures
D	Deliverable
DAG	Directed Acyclic Graph
DB	Database
DDS	Data Distribution Service
DID	Decentralized Identifier
DKMA	Distributed Key Management and Authentication
DLT	Distributed Ledger Technology
DoA	Description of Action
DP-ABC	Distributed Privacy Attribute-based Credentials
EAT	Entity Attestation Token
EC	European Commission
ESG	Edge Storage Gateway
ETL	Extract, Transform, Load
EUs	End Users
FCS	Fleet Control System
FL	Federated Learning
FPGA	Field Programmable Gate Arrays
GPU	Graphics Processing Unit
GUI	Graphical User Interface
HW	Hardware
IDS	Intrusion Detection System
IEC	International Electrotechnical Commission
IIoT	Industrial Internet of Things
IoC	Indicators of Compromise
IoT	Internet of Things
JWT	JSON Web Tokens
K8s	Kubernetes
KMS	Key Management System
KPI	Key Performance Indicator
M	Month
ML	Machine Learning
MQTT	Message Queueing Telemetry Transport
MTTR	Mean Time to Repair

NBS	Nash Bargaining Solution
NIR	Near-Infrared Spectrum
OCI	Open Container Initiative
OF	Operation Flow
OOM	Out-of-Memory
OT	Operational Technology
PDP	Policy Decision Point
PEP	Policy Enforcement Point
PMDS	Persistent Monitoring Data Storage
PoC	Proof of Concept
PPFL	Privacy-Preserving Federated Learning
PSM	Privacy and Security Manager
PVC	Persistent Volume Claim
QoS	Quality of Service
RAM	Random Access Memory
RDMA	Remote Direct Memory Access
REST	REpresentational State Transfer
RL	Reinforcement Learning
RLNC	Random Linear Network Coding
ROT	Resource Optimization Toolkit
SDK	Service Development Kit
SLA	Service Level Agreement
SOC	Soil Organic Carbon
SSI	Self-Sovereign Identity
SW	Software
TPU	Tensor Processing Unit
UAV	Unmanned Aerial Vehicles
UC	Use Case
UI	User Interface
URL	Uniform Resource Locator
VC	Verifiable Credentials
Vis-NIR	Visible and Near-Infrared Spectrum
VM	Virtual Machine
VP	Verifiable Presentations
VRAM	Video Random Access Memory
WAN	Wide Area Network
WP	Work Package
ZKP	Zero-Knowledge Proofs

Executive Summary

This deliverable presents the technical outcomes of Task 3.1: “Distributed Trust Management, Trust Propagation and Verification” and Task 3.2: “Data Management for Distributed Data Processing”, between M4 and M15.

First, the general EMPYREAN architecture is described, highlighting the role of each component developed in the two tasks. This allows the reader to contextualize each component’s purpose and place in the platform’s architecture.

This overview is followed by a detailed technical description of each component, including internal architecture, key workflows, public APIs, and planned integrations with other platform components. Five components are described, namely the Privacy and Security Manager, the DP-ABC Module, Cloud-native IoT device management, the Edge Storage Service, and the Decentralized and Distributed Data Manager. Beyond the technical descriptions, aspects related to the project’s management are discussed, continuing the work reported in D2.3 (M12). The relationship of each component with project objectives and KPIs is described. Furthermore, a brief description is included on how each component is going to be used and showcased by the project’s use cases.

Finally, future steps are highlighted as part of the conclusion. This includes a roadmap for the following phase of T3.1 and T3.2, including how these contributions will be reported.

1. Introduction

EMPYREAN targets a vision of application owners collaborating inside Associations, sharing resources and data across the edge-cloud continuum. A key enabler for making this vision a reality is a robust security and trust model for managing user applications, credentials and data across this highly distributed setting. The components of Work Package 3: "Security, Trust and Seamless Data and Computing Management" are given in this task.

This section provides a quick overview of this deliverable's content, comprising the purpose, document structure, and audience.

1.1 Purpose of this document

The present deliverable reports on the outcomes of Task 3.1: "Distributed Trust Management, Trust Propagation and Verification" and Task 3.2: "Data Management for Distributed Data Processing", during the first 12 months, between M4 and M15. Four partners were involved directly in these efforts: UMU, NUBIS, CC, and ZSCALE. The work follows from Work Package 2, which collected the project's requirements and established the general EMPYREAN architecture.

As one of the first technical deliverables of the project, the purpose of this deliverable is to describe how the components developed in these tasks fit into the general architecture of the project and provide their detailed description. Beyond the static view, where applicable, typical workflows are also described, providing a dynamic view. A connection to both project objectives, KPIs, and the use cases is established, laying the groundwork for the outcome validation and evaluation that will be performed in Work Package 6.

The document includes the initial API definitions and the integration plans between the various components, which are important steps in the integration work that will be performed in Work Package 5. As such, this deliverable is crucial in providing technical guidance to both the developers of the other platform services and the use case providers.

The writing of this deliverable coincided with the news that the partner providing the third use case would not be able to continue with their work in the project. The remaining consortium members have begun preparations and planning to address this unforeseen challenge, results will be reported in future deliverables.

The final report presenting the outcomes of T3.1 and T3.2 in the second phase of these tasks will be reported in Deliverable 3.3[1] in M26. Whereas the current deliverable focuses on the

[1] Deliverable 3.3: Final report on security, trust, seamless data and computing management. Due March 2026

technical specifications of each individual platform component, D3.3 will provide more details on integrations and APIs.

1.2 Document structure

The present deliverable is split into eight major chapters, centred around the components developed in Task 3.1 and Task 3.2:

- Introduction
- EMPYREAN Architecture mapping
- Privacy and Security Manager
- DP-ABC Module
- Cloud-native IoT device management
- Edge Storage Service
- Decentralized and Distributed Data Manager.
- Conclusion

1.3 Audience

This document is publicly available and should be of use to anyone interested in the description of the security, trust and data management aspects of EMPYREAN. It includes the initial description of the aforementioned components, their internal architecture, and the preliminary interfaces. Moreover, this document can also be useful to the general public for obtaining a better understanding of the framework and scope of the EMPYREAN project.

2 EMPYREAN Architecture Mapping

The EMPYREAN architecture was first introduced in deliverable D2.2[2], and later refined in its final version in D2.3[3]. This refinement incorporated key insights gained from the initial implementation phase. D2.3 provides a comprehensive overview of the architecture, detailing the EMPYREAN components, their interfaces, and the supported operational flows.

In this section, we present a concise description of the architecture (Figure 1) to support the discussion of the initial developments in WP3, particularly focusing on (i) mechanisms to provide trust management and privacy-preserving data access and confidentiality across the Associations, (ii) distributed, self-managed and encrypted hybrid data storage service for the continuum, (iii) decentralized and distributed data distribution service, and (iv) trusted computing entities to ensure secure boot and trusted execution across the Associations.

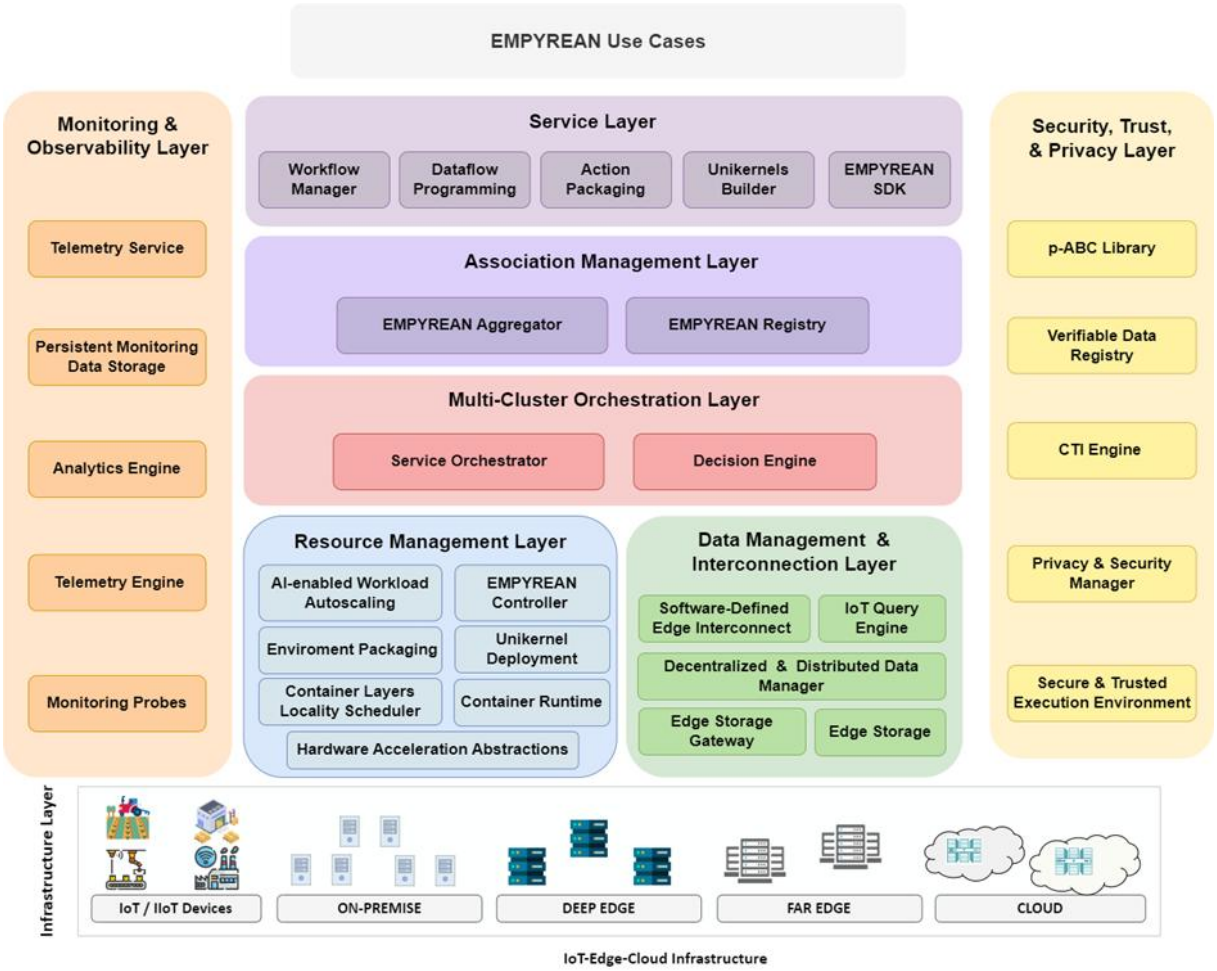


Figure 1: EMPYREAN high-level architecture

[2] D2.2: Initial Release of EMPYREAN Architecture, August 2024
[3] D2.3: Final EMPYREAN architecture, use cases analysis and KPIs, January 2025

In the context of the EMPYREAN Architecture, the ***Service Layer*** facilitates the development of Association-native applications by providing robust support for application-level adaptability, interoperability, elasticity, and scalability across the IoT-edge-cloud continuum. It addresses several key aspects, including: (a) the design and management of workflows for hyper-distributed applications, (b) cloud-native unikernel application development, and (c) data-flow description. A detailed design and initial implementation description of this layer's components is provided in deliverable D4.1[4] (M15).

The ***Association Management Layer*** dynamically manages Associations within the IoT-edge-cloud continuum. By forming resource federations, it enables seamless collaboration, resource sharing, and data distribution across various segments within the continuum. In conjunction with the Multi-Cluster Orchestration Layer, it plays a central role in EMPYREAN's distributed and autonomous management framework, establishing a resilient and adaptive Association-based continuum.

The ***Multi-Cluster Orchestration Layer*** handles service orchestration and resource management across EMPYREAN's disaggregated infrastructure. Leveraging autonomous, distributed decision-making mechanisms, it orchestrates hyper-distributed applications and enables self-driven adaptations. Multiple instances of this layer's components provide decentralized operations, optimized resource utilization, and scalability, while also supporting energy efficiency, fault tolerance, and high service quality. The design and development of the components for this and the Association Management layer are detailed in D4.2[5] (M15).

The ***Resource Management Layer*** unifies the management of IoT, edge, and cloud platforms under the EMPYREAN platform. It integrates software mechanisms for both platform-level scheduling (e.g., EMPYREAN Controller, AI-enabled Workload Autoscaling) and low-level operations (e.g., Unikernel Deployment). Operating within Kubernetes or K3s clusters, this layer is highly modular, simplifying the integration of new hardware and software components. Deliverables D3.2[6] (M15) and D4.1 (M15) presents the initial developments for this layer components.

The ***Data Management and Interconnection Layer*** ensures secure, scalable, and dynamic data communication and storage between IoT devices and computing resources. Operating at both cluster and Association levels, it integrates distributed data management mechanisms that enable seamless interaction between IoT, edge, and cloud environments. The novel data management mechanisms developed within the EMPYREAN are detailed in this deliverable, while the software-defined interconnection framework is presented in deliverable D3.2 (M15), and the IoT Query Engine in D4.1 (M15).

[4] D4.1: Low-code application description, seamless deployment and analytic-friendly distributed storage, April 2025
[5] D4.2: Intelligent resource management, cyber threat intelligence and EMPYREAN, April 2025
[6] D3.2: Software-defined edge interconnect and service assurance mechanisms, April 2025

The *Edge Storage Gateway* (Section 6.4.1) and *Edge Storage* (Section 6.4.4) are the foundations of EMPYREAN's secure and efficient edge storage service. These components manage storage across edge and cloud infrastructures, supporting hybrid policies for data placement, redundancy, and protection. They also utilize erasure coding techniques to ensure reliability and data integrity. Additionally, the *Decentralized and Distributed Data Manager* (Section 7) provides scalable, decentralized, and distributed communication mechanisms with efficient publish/subscribe and data querying capabilities. It facilitates communication between IoT devices and edge computing and storage resources across various providers/administrative domains, connectivity types (e.g., extremely constrained networks), technologies, and network zones.

The **Infrastructure Layer** comprises heterogeneous resources distributed across multiple administrative and technological domains including, (i) IoT/IIoT devices, robots, and on-premise edge resources where data is generated and service requests initiated, (ii) deep and far-edges, close and further from the end users/devices, for real-time processing and aggregation, and (iii) federated multi-cloud environments for enhanced robustness, cost-efficiency, and vendor independence in data storage and replication.

The architecture is complemented by the Security, Trust, and Privacy Layer and the Monitoring and Observability Layer, which are across the other layers, providing critical functionalities for the overall platform.

The **Security, Trust, and Privacy Layer** integrates distributed components to ensure secure access, privacy-preserving operations, and trusted execution across the platform. Operating at both cluster and Association levels, it establishes secure execution environments where trust relationships between data-generating and data-processing entities are continuously verified through distributed trust mechanisms. In parallel, identity and data access management components ensure controlled access and data confidentiality among different entities. This deliverable presents the core components of this layer, while the Cyber Threat Intelligence (CTI) Engine is detailed D4.2 (M15).

The *Privacy and Security Manager* (Section 3) and *DP-ABC* library (Section 4) offer robust identity and access management alongside attribute-based credential management. The Privacy and Security Manager ensures secure and private identity management, data verification, and a strong cryptographic foundation for managing privacy-preserving attribute-based credentials across the platform. The DP-ABC library complements this by offering a distributed privacy-preserving attribute-based credential system based on PS multi-signatures. Moreover, the *Secure and Trusted Execution Environment* (Section 5) establishes secure and trusted execution across the IoT-edge-cloud continuum, supporting secure and measured boot mechanisms. It enables applications to be deployed seamlessly with varying levels of security and trustworthiness across different hardware platforms. This allows for scalable and transparent operation, from micro deep edge devices to the far edge and cloud environments, ensuring that security requirements are met at all levels.

The ***Monitoring and Observability Layer*** provides real-time monitoring, observability, and service assurance through distributed telemetry mechanisms and data-driven analytics. It dynamically collects and analyses a wide range of metrics across heterogeneous infrastructures and deployed applications, ensuring system health, performance, and availability. These insights support automated control and optimization. Telemetry components are detailed in deliverable D4.2 (M15), with the service assurance mechanisms covered in D3.2 (M15).

3 Privacy and Security Manager

3.1 Overview

The Privacy and Security Manager (PSM) is a core component of the EMPYREAN platform, ensuring trust, privacy, and security across the IoT-edge-cloud continuum. It provides decentralized identity management, verifiable credentials (VCs), secure authentication, and policy enforcement. The PSM enables dynamic and decentralized security policies, ensuring seamless interactions between devices, users, and services in EMPYREAN Associations.

3.2 Features

The PSM offers:

- Self-sovereign identity (SSI) management using Decentralized Identifiers (DIDs) and Verifiable Credentials (VCs).
- Secure authentication via DID-signed JWTs (JSON Web Tokens).
- Dynamic policy enforcement with blockchain-based smart contracts.
- Zero-Knowledge Proofs (ZKPs) for privacy-preserving authentication.

3.3 Novelty - Beyond the State of the Art

Unlike traditional access control models that rely on centralized identity providers, the Privacy and Security Manager (PSM) leverages blockchain technology to distribute trust among multiple parties. By integrating Zero-Knowledge Proofs (ZKPs) and Decentralized Identifiers (DIDs), it enhances security without sacrificing privacy. The ability to dynamically update security policies via smart contracts makes it more adaptable than conventional security frameworks.

3.4 Relation to project objectives and KPI

The PSM directly supports EMPYREAN's goals of ensuring security, privacy, and multi-party trust within distributed environments.

It contributes to several key performance indicators (KPIs):

Table 1: EMPYREAN Technical KPIs related to the Privacy and Security Manager

ID	Indicator	Success Criteria	Objective
T3.1	Number of trustworthy identity and trust management processes enabled by smart contracts.	>=3	3

T3.2	Accuracy of user and device verification and authentication.	> 99%;	3
T3.3	Reduction of privacy violation incidents in data sharing.	> 50%;	3

The component is relevant in achieving several project objectives:

- F_ASSOC.2, F_ASSOC.3, F_ASSOC.5
- F_ST.1, F_ST.2, F_ST.3, F_ST.6
- F_GR.1, F_GR.2

These cover a range of topics from storage, to the secure operation of Associations.

3.5 Architecture and Implementation

3.5.1 Internal Component Architecture

The Privacy and Security Manager (PSM) is designed as a modular, containerized microservice that integrates seamlessly into the decentralized EMPYREAN ecosystem. Its architecture embraces the Association-based continuum model, ensuring that each instance operates autonomously within an Association while maintaining interoperability across the broader IoT-edge-cloud fabric.

Deployed within Kubernetes clusters, the PSM leverages modern orchestration practices to ensure fault-tolerant, high-availability operations. It interfaces with the EMPYREAN Aggregator and Service Orchestrator to enforce access policies and identity verification in real time. Through tight integration with Distributed Ledger Technologies (DLTs), it supports immutable policy storage and decentralized identity management.

3.5.2 Authentication and Authorization Module

At the heart of the PSM lies the DID-based authentication framework. This module utilizes the W3C-compliant Decentralized Identifier (DID) model to support cryptographically verifiable identities. When users or devices interact with EMPYREAN services, their access requests are validated through DID-based signatures, which are then encapsulated in JSON Web Tokens (JWTs) for fast and secure API interactions.

This module ensures:

- Stateless authentication across services via signed JWTs.
- Integration with Hyperledger Aries and Fabric for DID registration and resolution.
- Revocation checking and trust scoring mechanisms.

3.5.3 Verifiable Credential Management

The PSM functions as both an issuer and verifier of Verifiable Credentials (VCs) and Verifiable Presentations (VPs). Using Privacy-preserving Attribute-Based Credentials (p-ABC) and Zero-Knowledge Proofs (ZKPs), it allows selective disclosure of identity attributes, enabling:

- Minimal disclosure and user consent enforcement.
- Dynamic, context-aware access policies.
- Interoperability with third-party VC issuers via a Trusted Issuer Registry.

3.5.4 Policy Enforcement Engine - XACML

To guarantee fine-grained access control, the PSM includes an XACML (eXtensible Access Control Markup Language)-compatible Policy Enforcement Engine. Policies are defined in a declarative manner and deployed via smart contracts to a DLT, ensuring both transparency and immutability.

This component provides:

- Runtime evaluation of access requests against stored policies.
- Blockchain-backed auditability of enforcement decisions.
- Coordination with PDP and PEP microservices for distributed enforcement.

3.5.5 Registry, Blockchain & Smart Contracts

Verifiable Data Registry

- **Trusted Issuers List**: Maintains validated issuers of credentials to enforce trust boundaries.
- **Trusted Participant List**: Stored on the blockchain, it includes DIDs of Association members alongside their trust scores and interaction history.

Blockchain & Smart contracts

The Privacy and Security Manager (PSM) leverages **Hyperledger Fabric** to ensure decentralized, tamper-proof management of identity, policy, and access decisions. Several smart contracts have been deployed to support these functions:

- Custom extensions of Fabric's chaincode handle the creation and retrieval of **Decentralized Identifiers (DIDs)**, enabling secure identity resolution across EMPYREAN entities.

- **Policy Rule Storage**: Access control policies defined in **XACML** are persistently stored on the blockchain via dedicated smart contracts. This ensures that policy rules are **transparent**, **auditable**, and **immutable**, supporting consistent enforcement across distributed environments.
- **Access Traceability**: A dedicated smart contract logs all **access control decisions** (e.g., "GRANTED", "DENIED") along with metadata such as timestamps, resources accessed, and associated DIDs. This historic logging mechanism supports **compliance auditing**, **forensic investigation**, and dynamic policy refinement based on real usage data.

3.5.6 OP-TEE Module

To support secure and tamper-resistant execution of privacy-critical operations, the Privacy and Security Manager integrates **OP-TEE** (Open Portable Trusted Execution Environment) as a trusted module. OP-TEE provides a **hardware-assisted secure execution context** based on ARM TrustZone technology, establishing a dual-world architecture: the **Normal World** (Linux OS) and the **Secure World** (TEE).

Key features and architectural roles include:

- **Secure Execution Environment**: OP-TEE enables the isolation of sensitive operations such as **cryptographic key handling**, **Verifiable Credential generation**, and **policy decision execution**, reducing the attack surface and preventing confidential data leakage.
- **Trusted Applications (TAs)**: Security-critical logic runs within **Trusted Applications**, executed exclusively in the Secure World. These TAs are independently signed and verified, ensuring code integrity and authenticity at runtime.
- **ARM TrustZone Integration**: Hardware-based security is enforced through ARM TrustZone, which separates memory, processing, and peripherals between the secure and normal execution environments.
- **Client–TEE Communication**: Applications in the Normal World (Client Applications or CAs) interact with TAs via the **TEE Client API Library**, which communicates securely with the **OP-TEE driver**. The driver manages context switches and data marshalling between the worlds.
- **TEE Supplicant Interface**: For certain operations (e.g., file system or crypto access), the **TEE Supplicant** provides a bridge between the Normal World services and the Secure World, without compromising TEE isolation.

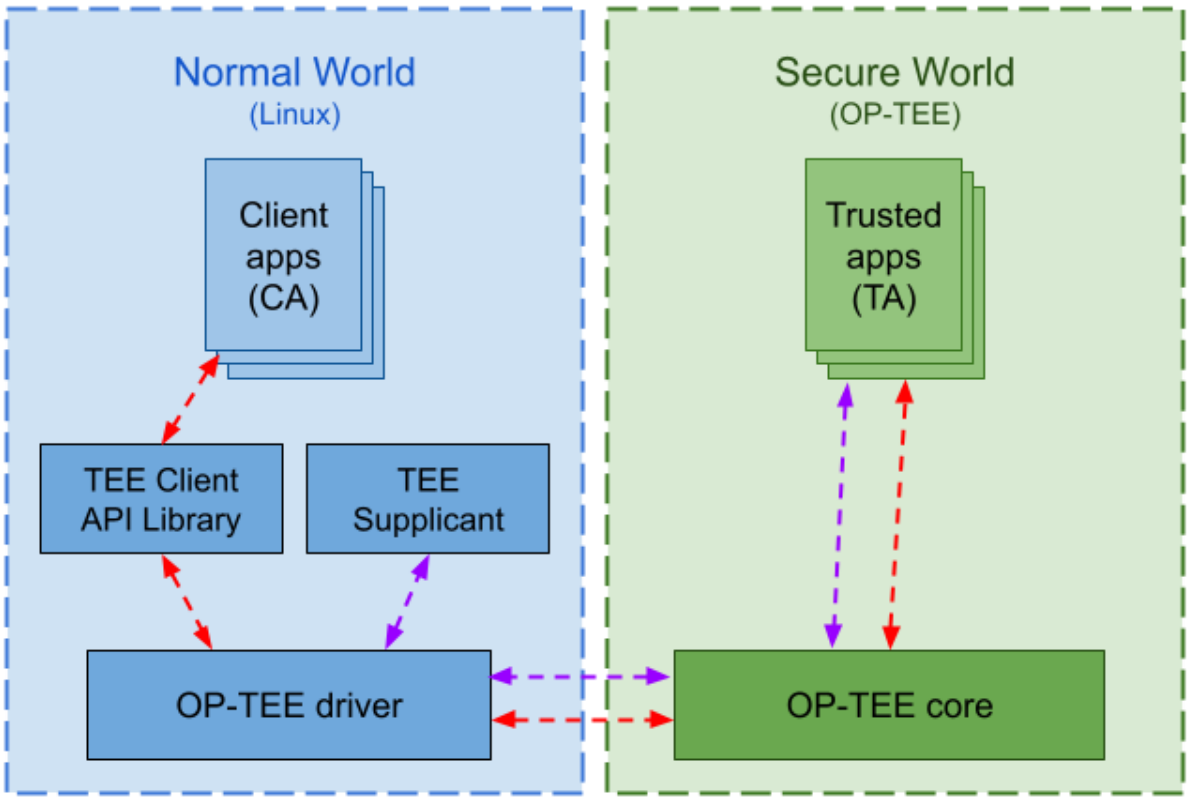


Figure 2: OP-TEE Dual-world model

Figure 2 illustrates the dual-world model of OP-TEE, showcasing the separation and secure communication flow between the Client Applications and Trusted Applications.

This integration ensures the **confidentiality, integrity, and isolation** of critical security functions in the EMPYREAN PSM, aligning with the project's objective to establish secure, trustworthy environments within each IoT-edge Association.

3.5.7 Deployment

The deployment of the privacy and security manager is available building the service in every EMPYREAN Entity as a docker-compose service with all the containers running the agents and fabric nodes.

3.5.8 Workflows

Combined Enrolment and Authentication/Authorization Workflow

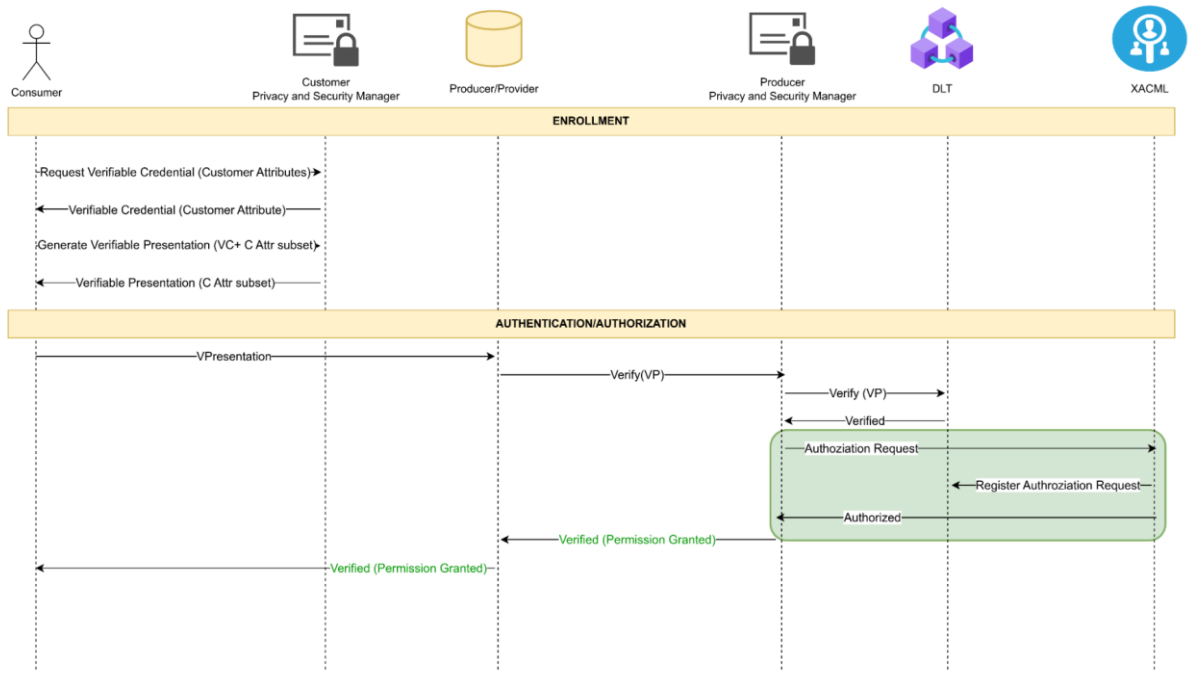


Figure 3: Enrollment and AuthN/AuthZ PSM Workflow

The EMPYREAN Privacy and Security Manager (PSM) supports a seamless workflow that integrates **identity enrolment** with **authentication and authorization**, ensuring that entities are onboarded securely and evaluated dynamically at each interaction. This combined flow is critical to maintaining zero-trust principles while enabling interoperability and privacy-preserving access.

The sequence is depicted in Figure 3 and involves the following phases:

1. Enrolment Phase

- **Credential Request**:
  The **consumer** (e.g., a user, device, or service) initiates the enrolment process by requesting a **Verifiable Credential (VC)** from its local PSM. This VC contains a structured set of attributes, such as identity details, role, or contextual information.
- **Credential Issuance**:
  The **Customer PSM** validates the request and issues a digitally signed VC, possibly backed by a decentralized identity (DID). This credential can later be verified by any EMPYREAN participant based on the issuer's trust level.
- **Presentation Preparation**:
  To preserve privacy, the consumer selects a subset of attributes from the VC to

generate a **Verifiable Presentation (VP)**. This allows for **selective disclosure**, where only the minimum required attributes are shared (e.g., using ZKPs or p-ABC schemes).

2. Authentication and Authorization Phase

- **VP Submission**:
  The VP is sent from the consumer to the **Producer/Provider PSM** as part of an access request.
- **Verification**:
  The producer-side PSM verifies the VP's:
  - **Cryptographic integrity** (e.g., signature validation),
  - **Trustworthiness of the issuer**, using a blockchain-based **Trusted Issuer Registry**, and
  - **Validity of the disclosed attributes**.
- **Authorization Request**:
  Once verified, the PSM forwards an authorization request to the internal **XACML-based Policy Enforcement Engine (PEP/PDP)**. This engine evaluates whether the presented attributes satisfy the access control rules defined in the active policies.
- **Traceability and Audit Logging**:
  The authorization request is **registered on the DLT**, ensuring immutable traceability for compliance and post-event analysis.
- **Decision and Feedback**:
  If the policy conditions are met, the access is **authorized** and permission is granted to the consumer. This decision is propagated back to confirm that the consumer is successfully authenticated and authorized.

The PSM is deployed as a **containerized microservice** within each EMPYREAN **Association**. It integrates with the **EMPYREAN Aggregator** for access control and policy enforcement. The **DLT-based policy management** allows for real-time security updates across distributed environments.

For scalability, the PSM **leverages Kubernetes-based orchestration**, ensuring redundancy and high availability. It interacts with **Hyperledger Fabric** for **secure identity verification** and **policy enforcement**.

The implementation also includes **RESTful APIs** to facilitate interoperability with external services.

3.6 Public APIs

The PSM exposes REST APIs for authentication, credential management, policy enforcement, and JWT signing. Below is a high-level overview of the available API endpoints:

Identity and Credential Management

- **/VerifyCredential** (POST) - Verifies a **Verifiable Credential (VC)**.
- **/generateDID** (POST) - Generates a **Decentralized Identifier (DID)**.
- **/generateVp** (POST) - Creates a **Verifiable Presentation (VP)**.
- **/getVCredential** (POST) - Retrieves a stored **VC** for an entity.

Enrollment and Trust Management

- **/acceptEnrolment** (POST) - Checks if an enrolment request is valid.
- **/doEnrolment** (POST) - Submits a **VC** for enrolment.
- **/trustedIssuers** (GET) - Fetches a **list of trusted issuers**.

JWT Management

- **/signJWTContent** (POST) - Digitally **signs a JWT**.
- **/verifyJWTContent** (POST) - Verifies the **validity of a JWT**.

TEE

- **/tee/generatekeypair** (POST) - Generate keypair in a Trusted Execution Environment

These APIs enable secure and seamless authentication, access control, and trust management across EMPYREAN services.

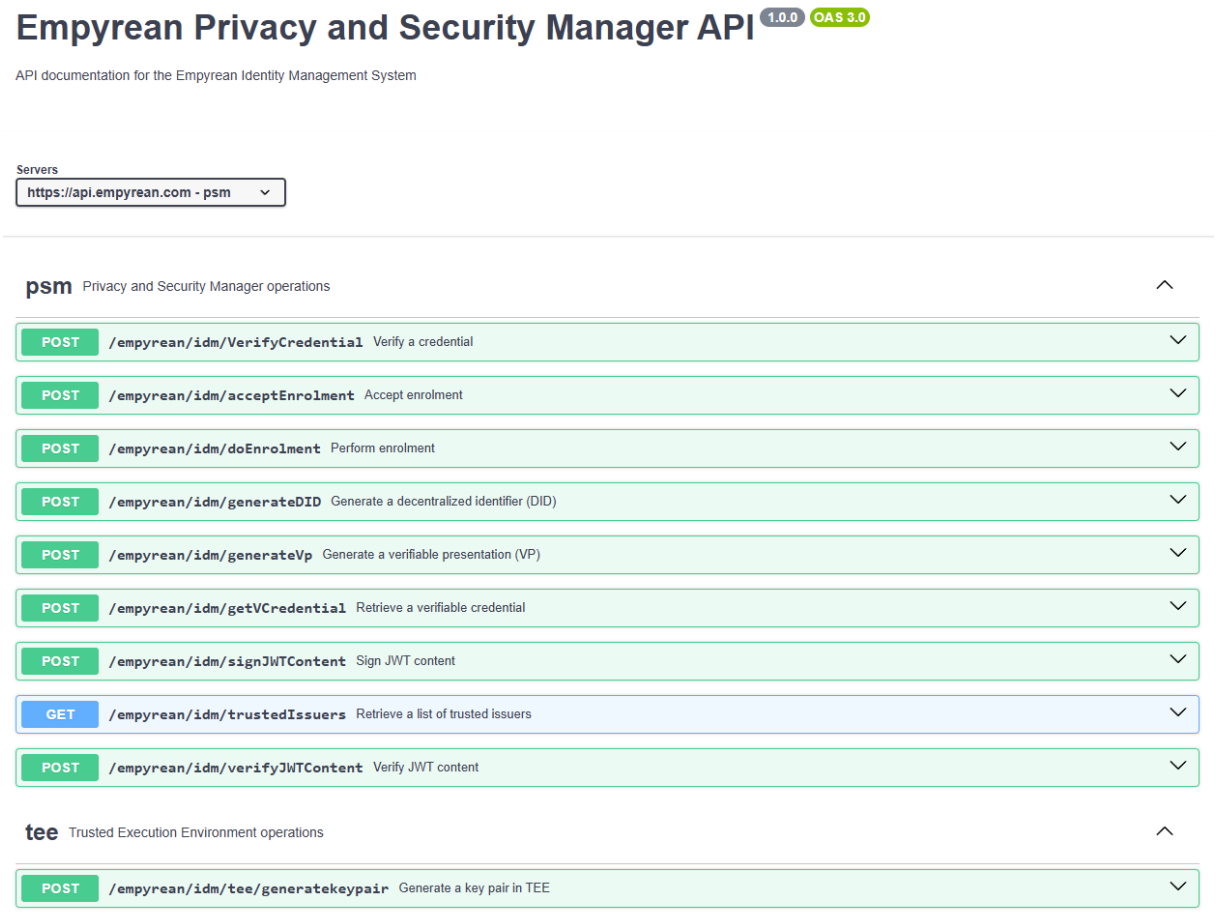


Figure 4: Privacy and Security Manager API

3.7 Integration with EMPYREAN Platform services

The PSM is tightly integrated into the EMPYREAN ecosystem, ensuring that all security policies are dynamically enforced while maintaining a privacy-preserving environment.

EMPYREAN Aggregator

Works as the central orchestrator for identity verification and policy enforcement, the aggregator will be the proxy between services and users and he will manage the access with the PDP/PEP Proxy of his PSM.

Edge Storage Gateway

The **Edge Storage Gateway (ESG)**, a key component of Chocolate Cloud's **Edge Storage Service (ESS)**, plays a dual role in EMPYREAN: it not only facilitates high-performance, hybrid edge-cloud object storage, but also contributes to the **identity, access control, and privacy infrastructure** by acting as a **source of access proofs** for credential-based security mechanisms managed by the PSM.

This integration bridges secure storage access with **self-sovereign identity (SSI)** principles, enabling users to control and selectively disclose identity attributes based on storage-related capabilities and policies.

Core Concept

The ESG verifies users' authenticated access to edge storage services and exposes this verification to the user as a **verifiable proof**. This proof becomes the basis for issuing **Verifiable Credentials (VCs)** against a trusted issuer, which can later be used for **privacy-preserving authorization** workflows across the EMPYREAN platform.

Workflow Overview

1. **Storage Access and Identity Proofs**
   Users interact with the **Edge Storage Gateway (ESG)** using standard S3-compatible APIs. Upon successful authentication (e.g., through API keys or pre-signed URLs), the ESG can generate **signed proofs** reflecting the user's verified access to specific storage providers and resource policies.

2. **Credential Issuance via PSM**
   These proofs are forwarded to the **Privacy and Security Manager**, which present them to a trusted issuer for credential issuance. This VC could contain storage-related attributes such as:

   - Proven access to cloud services
   - Supported operations (e.g., read/write access)
   - Encryption or erasure coding capabilities
   - Location-awareness or policy alignment (e.g., edge-only, hybrid, GDPR-compliant)

3. **Selective Disclosure with p-ABCs and ZKPs**
   When interacting with a relying party or accessing sensitive data, the user generates a **Verifiable Presentation (VP)** derived from the VC. Using **privacy-preserving attribute-based credentials (p-ABCs)** and **zero-knowledge proofs (ZKPs)**, the user can **disclose only the necessary attributes** required for authorization (e.g., "has access to encrypted

edge storage" without revealing which provider or key structure).

4. **Authorization and Policy Enforcement**
   The VP or JWT Derived is evaluated by the relying service (e.g., an ML pipeline orchestrator or distributed application component) against policies defined in the **PSM's XACML-compatible engine** and anchored in the **blockchain**. If the attributes match the policy, access is granted; otherwise, the request is denied or flagged.

OP-TEE Module and P-ABC

To protect sensitive operations and cryptographic material in the EMPYREAN ecosystem at a high level of security, the **Privacy and Security Manager (PSM)** integrates with **OP-TEE**, a Trusted Execution Environment (TEE) that leverages **ARM TrustZone** to create a secure, isolated processing domain.

As illustrated in Figure 5, the PSM uses OP-TEE to delegate security-critical tasks while maintaining strict separation between the normal Linux-based OS and the secure world:

- **Secure Cryptographic Operations**
  The PSM offloads cryptographic computations (e.g., signing, decryption, key handling) to OP-TEE. These operations are isolated from the normal execution environment to prevent leakage or tampering.
- **Trusted Execution**
  Inside OP-TEE's Secure World, **Trusted Applications (TAs)** perform the requested tasks. This environment ensures that sensitive data never leaves the protected memory space, even in the presence of compromised applications in the Normal World.
- **API Communication**
  The PSM interacts with OP-TEE via a secure API. Requests are passed through the OP-TEE Client Library and driver in the Normal World into the Secure World where they are handled securely.
- **Enhanced Privacy and Security**
  This architecture ensures end-to-end protection of sensitive operations. Even if the operating system or application layer is compromised, secrets managed by the PSM remain protected by the TEE.

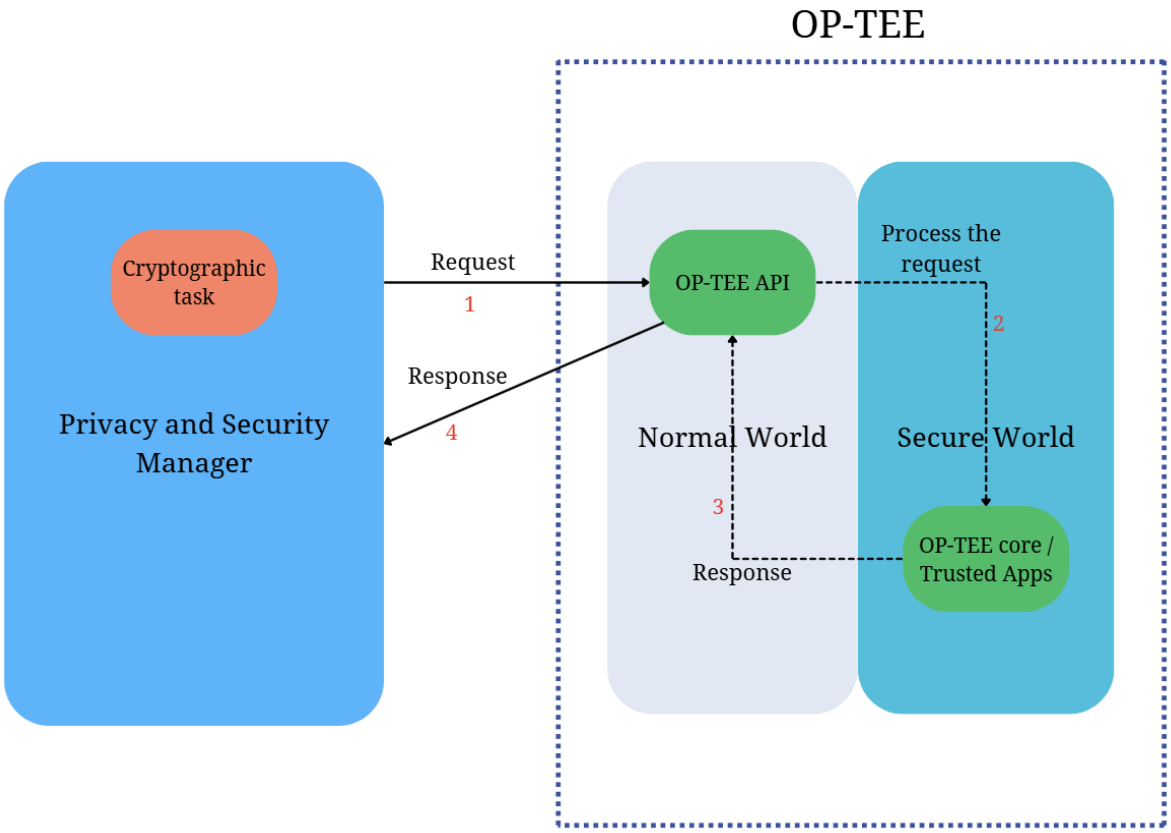


Figure 5: Overview of integration between OP-TEE module and P-ABC

Figure 5 illustrates how the **Privacy and Security Manager (PSM)** securely delegates sensitive cryptographic tasks to the **OP-TEE Trusted Execution Environment**, following a four-step process across the **Normal World** and **Secure World** domains.

1. Request Initiation

The PSM identifies a cryptographic task (e.g., signature generation or private key operation) that must be executed in a secure environment. It sends a **request** to the **OP-TEE API**, located in the Normal World (Linux-based user space). This request includes the parameters or data required to perform the operation.

2. Secure World Processing

The OP-TEE API passes the request through the secure communication channel to the **OP-TEE core**, which resides in the Secure World. The request is then routed to the appropriate **Trusted Application (TA)** capable of handling the task. Inside the Secure World, the TA performs the cryptographic operation in isolation from the main OS, in our case, generating keypairs with the P-ABC Module.

3. Response Transfer

Once the TA completes the task, it generates a **response**, which is securely returned from the Secure World to the OP-TEE API in the Normal World. This response may include, for example, a signed message or encrypted result.

4. Result Delivery to PSM

Finally, the OP-TEE API sends the result back to the PSM. The PSM can now proceed with its workflow (e.g., issuing a credential, authorizing access, etc.) using the securely computed output.

Ryax Workflow Manager

To enhance interoperability and trust management across the EMPYREAN platform, the **Privacy and Security Manager (PSM)** can integrate with external identity and attribute providers. One such integration involves using **RYAX's Keycloak** as a bridge to provide certified identity attributes to EMPYREAN entities.

As illustrated in Figure 6, this setup enables the PSM to issue **privacy-preserving Attribute-Based Credentials (p-ABCs)**, while leveraging externally validated attributes for enrollment and authorization purposes.

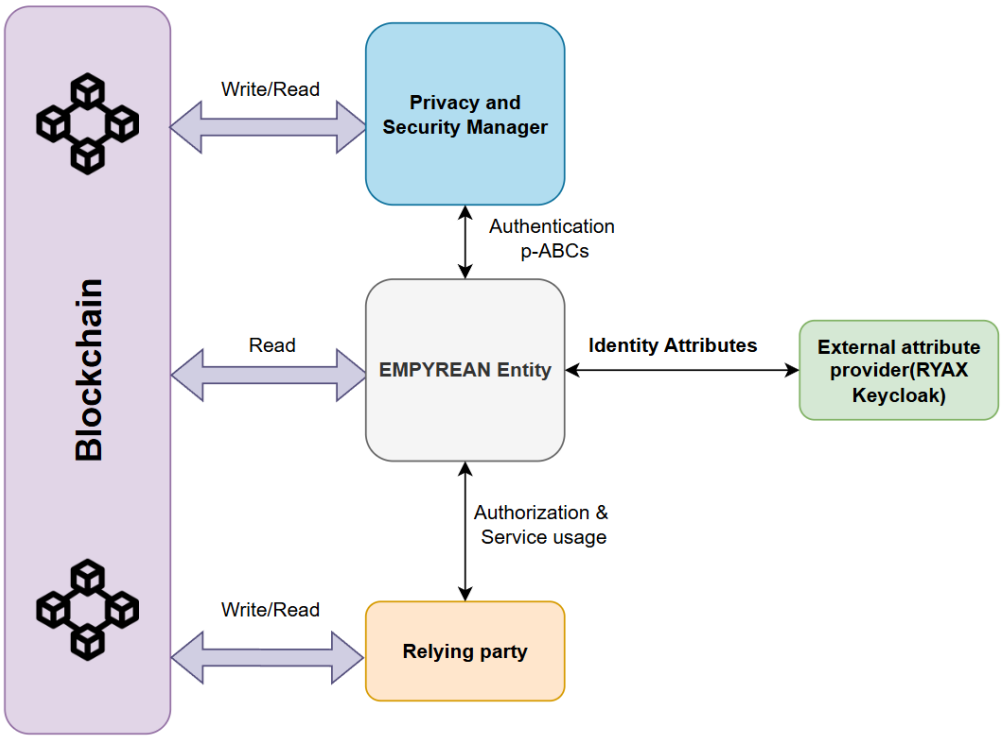


Figure 6: Overview of integration between PSM and Ryax Workflow Manager

- **Privacy and Security Manager (PSM)**
  Responsible for authenticating entities and issuing **Verifiable Credentials (VCs)** in the form of p-ABCs. These credentials enable users to later generate Verifiable Presentations for minimal-disclosure authentication and access control.
- **External Attribute Provider (RYAX Keycloak)**
  Acts as a trusted **issuer of identity attributes**, such as roles, group memberships, or usage entitlements. These attributes are used by the PSM to construct VCs during the

enrollment phase. Keycloak provides a standards-compliant interface for attribute assertions, supporting OpenID Connect and SAML.

- **EMPYREAN Entity**
  A component (e.g., device, user, or microservice) operating within the EMPYREAN Association. It receives p-ABCs from the PSM and uses them to create unlinkable tokens or Verifiable Presentations, enabling **selective disclosure** of attributes depending on the access context.
- **Relying Party**
  Any EMPYREAN platform service or component that **enforces Attribute-Based Access Control (ABAC)**. Relying parties evaluate incoming VPs against defined access policies and interact with the blockchain to verify identity assertions, audit logs, or permissions.
- **Blockchain Layer**
  Serves as a trusted ledger for storing access policies, audit logs, trusted issuers, and verification events. All key actors interact with the blockchain for transparency and traceability:
    - The PSM **writes/reads** identity credentials and trust status.
    - EMPYREAN Entities **read** validation rules or credential status.

Relaying Parties **log** authorization decisions and read policies.

3.8 Relation to use cases

The PSM plays a crucial role in EMPYREAN's **core use cases (UCs)** by securing device interactions and enforcing **trusted access**.

UC1: Secure Robotic Operations in Manufacturing

- Ensures that **only authorized robots** access machining data.
- Verifies identity using **VCs and DIDs** before allowing data exchange.
- Traceability for industrial operation with DID's matching

UC2: Privacy-Preserving Agriculture Monitoring

- Protects **sensor data** from unauthorized access.
- Uses **ZKPs** to allow farmers to share specific data with researchers.

4 DP-ABC Module

4.1 Overview

The Distributed Privacy Attribute-based Credentials (**DP-ABC) Library** is a cryptographic library designed to provide **privacy-preserving attribute-based credentials (ABC)**. It enables **IoT and low-power edge devices** to prove attributes selectively while ensuring security and minimal computational overhead. The library implements **Zero-Knowledge Proofs (ZKPs), BBS signatures, and attribute hiding** to enable **secure, privacy-focused authentication and access control** in distributed environments.

4.2 Features

- **Decentralized Credential Issuance** – Enables devices to generate and manage **attribute-based credentials (VCs)** without relying on a centralized authority.
- **Selective Disclosure** – Users/devices can reveal only necessary attributes without exposing full identity.
- **Zero-Knowledge Proofs (ZKPs)** – Allows authentication without disclosing sensitive information.
- **BBS Signature Scheme** – Supports **multi-message signing** and **unlinkable credentials**.
- **Standalone Operation for IoT Devices** – Low-computation IoT devices can use the **DP-ABC library independently**, without the **Privacy and Security Manager (PSM)**, to generate and verify credentials locally.

4.3 Novelty - Beyond the State of the Art

Current credential systems often **require cloud-based verification**, which can **increase latency and reduce privacy**. The **DP-ABC Library** enables **standalone credential verification on low-power IoT devices**, making it an ideal choice for **EMPYREAN's decentralized and device-centric architecture**. It is optimized for **offline authentication**, ensuring secure interactions even in constrained environments.

4.4 Relation to Project Objectives and KPIs

The DP-ABC Library plays a core role in EMPYREAN’s vision of a decentralized, secure, and autonomous IoT-edge-cloud environment. It aligns with key project objectives:

- Privacy-Preserving Authentication – Supports Zero-Knowledge Proofs to prevent unnecessary data exposure.
- Scalability in IoT and Edge Networks – Enables low-power devices to handle identity verification without relying on cloud services.

- Secure Device Interactions – Contributes to EMPYREAN's KPI of reducing privacy violations in data sharing by over 50%.
- Autonomous IoT Security – Allows devices to verify credentials independently, enhancing resilience in disconnected or remote environments.

4.5 Architecture

The **DP-ABC Library (**Figure 7**)** is designed as a **lightweight, embeddable library** that integrates **seamlessly into IoT devices, edge nodes, and high-performance systems**. It consists of the following modules:

- **BBS Signature Module:** Implements the **BBS signature scheme**, allowing devices to sign and verify multiple messages securely. This enables **unlinkable credential proofs**, preventing tracking and correlation of authentication requests.

- **Credential Management Module:** Handles **issuance, storage, and revocation** of attribute-based credentials (VCs). It allows IoT devices to **self-manage credentials** or rely on **trusted issuers**.

- **Zero-Knowledge Proof Generation Module:** Generates **cryptographic proofs** that allow devices to prove **possession of credentials** without revealing unnecessary attributes. This ensures **secure and private interactions** between IoT devices and platform services.

Configuration and Integration

- **Build**: The library is built as a static/shared `.so` or `.a` object and can be linked in C/C++ applications.
- **Dependencies**: None external; uses its own wrapper over EC cryptographic primitives.
- **Integration Target**: It is being integrated into the Privacy and Security Manager (PSM) and used to:
    - Sign JSON Web Tokens (JWTs) using DIDs.
    - Generate **Verifiable Presentations (VPs)** from **Verifiable Credentials (VCs)**.
    - Apply Zero-Knowledge Proofs for **selective disclosure**.

Implementation Details

- **Credential Issuance**:
    - Supports generation of multi-attribute credentials signed by trusted authorities.
- **Credential Presentation**:
    - End-users can present subsets of their credentials to verifiers, using non-interactive ZKPs.

- **Security Features**:
  - Based on **pairing-friendly curves**, the library ensures unforgeability and unlinkability of credentials.
  - Fully supports **offline verification** (i.e., credentials can be verified without querying the issuer).

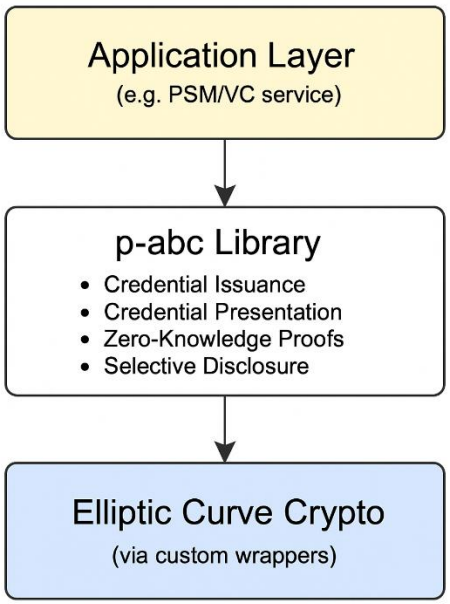


Figure 7: DP-ABC architectural overview.

4.6 Implementation

The DP-ABC Library is implemented in C and optimized for embedded devices and high-performance systems. It provides:

- Cross-Platform Compatibility – Runs on IoT microcontrollers, edge devices, and standard Linux/Windows environments.
- Cryptographic Performance Optimization – Utilizes hardware-accelerated cryptographic libraries where available.
- Minimal Resource Footprint – Designed to consume minimal CPU and memory, ensuring efficient execution on low-power devices.

4.7 Public APIs

The DP-ABC library provides a C-based API designed to support decentralized, privacy-preserving credential issuance and verification. Below is a summary of its key public functions, which constitute the core interface for applications integrating the library (e.g. the EMPYREAN Privacy and Security Manager).

- **dpabc_init()**
  Initializes the internal library state and cryptographic parameters. This function must be called once before using any other library functionality.
- **dpabc_clear()**
  Cleans up allocated memory and cryptographic structures. Should be called at the end of the application to prevent memory leaks.
- **dpabc_issuer_keygen(...)**
  Generates a key pair (secret and public keys) for a credential issuer. The number of supported attributes is specified by the caller. This function is used to initialize trusted authorities in the ecosystem.
- **dpabc_user_commit(...)**
  Computes a cryptographic commitment over a user's attributes and secret key. This is the first step in the issuance protocol, where the user proves knowledge of attributes without revealing them.
- **dpabc_issue(...)**
  Issues a credential for a user by combining the issuer's secret key with the user's commitment and list of attributes. The result is a signed credential that the user can later present in a privacy-preserving way.
- **dpabc_prove(...)**
  Generates a **zero-knowledge proof** of possession of a valid credential. The user can specify which attributes (if any) to reveal in the proof, enabling **selective disclosure**.
- **dpabc_verify(...)**
  Verifies a zero-knowledge proof generated by a user. This includes checking the cryptographic validity of the proof and that any disclosed attributes match expectations.

These API calls enable the full lifecycle of **verifiable credentials**:

- Key generation for issuers
- Secure, private credential issuance
- Selective disclosure and privacy-preserving authentication

- Verification of identity or attribute claims without centralized databases

The API is designed for modular integration and is currently being embedded in the EMPYREAN platform's Privacy and Security Manager (PSM), where it supports decentralized identifiers (DIDs), Verifiable Credentials (VCs), and cryptographic access control mechanisms.

4.8 Integration with EMPYREAN Platform Services

The DP-ABC Library acts as a foundational component of the Privacy and Security Manager (PSM) but is also designed for standalone use by IoT devices.

For Low-Power IoT Devices:

- Devices can generate, store, and verify credentials independently.
- Eliminates the need for constant cloud connectivity.
- Reduces latency in security operations.

For PSM-Managed Authentication:

- The PSM can issue credentials to devices.
- IoT devices can submit ZKPs to the PSM for access control decisions.
- P-ABC can be deployed in a TEE to perform cryptographic operations.

4.9 Relation to Use Cases

The DP-ABC Library enhances security and privacy across all EMPYREAN Use Cases (UCs):

UC1: Secure Robotic Operations in Manufacturing

Robots can verify credentials locally, ensuring only authorized units interact with manufacturing processes.

UC2: Privacy-Preserving Agriculture Monitoring

Sensors can share anonymized environmental data without exposing location details. Farmers retain control over what data is disclosed.

5 Secure and Trusted Execution Environment

5.1 Overview

Within the EMPYREAN project, NUBIS introduces a secure, cloud-native orchestration and device management framework, purpose-built for the IoT–edge–cloud continuum. This framework is developed exclusively within EMPYREAN to address the unique challenges of managing highly constrained devices, and in particular those based on ESP32-family MCUs, through a modern, Kubernetes-native lens.

Additionally, NUBIS builds on our team's foundational expertise in lightweight virtualization and sandboxed container runtimes, which serve as a key enabling layer for secure and efficient workload isolation across heterogeneous environments. These runtimes, including minimal VMs, microVMs, and unikernel-inspired containers, are tailored to offer strong security boundaries and low overhead, making them ideal for edge and IoT deployments. As part of EMPYREAN, we enhance this technology with tighter Kubernetes integration, enabling fine-grained resource control, secure multi-tenancy, and seamless offloading across the compute continuum.

The cloud-native IoT stack developed within NUBIS leverages these enhanced runtimes alongside Kubernetes-native constructs such as CRDs and controllers. It provides secure device onboarding using Entity Attestation Tokens (EATs) and Unique Device Secrets (UDS), robust OTA firmware management via OCI-packaged firmware blobs, flashed using “flashjobs” and intelligent workload orchestration with AI inference offloading through the vAccel framework[7]. By integrating Akri[8] for dynamic device discovery and adopting privacy-preserving design principles, NUBIS enables secure, scalable, and compute-aware IoT deployments across EMPYREAN Associations.

5.2 Relation to Project Objectives and KPIs

This framework directly addresses several key EMPYREAN objectives, particularly those related to secure device integration, dynamic resource management, and efficient execution across the edge-cloud continuum.

Table 2: EMPYREAN Technical KPIs related to Security and Trust Management

ID	Indicator	Success Criteria	Objective
T3.2	Accuracy of user and device verification and authentication	>99%	3

[7] vAccel framework: https://vaccel.org
[8] Akri: https://docs.akri.sh

T3.3	Reduction of privacy violation incidents in data sharing	>50%	3
T3.4	Time reduction for read/write on edge vs. cloud	by 40%	3
T3.5	Ability to access data on the edge when cloud connectivity is lost	Demo	3
T4.1	Increase small-message transfer performance at application level	by 3x	4
T5.1	Reduce development time of continuum-native applications	>20% decrease vs. SoTA	5
T5.2	Number of supported hardware architectures	>3	5

These KPIs are met through the following contributions:

- Zero-touch onboarding and attestation based on EAT and UDS improve T3.2.
- Confidential OTA updates and device telemetry routing reduce privacy incidents (T3.3).
- Edge-based OTA delivery reduces dependence on the cloud (T3.4, T3.5).
- Optimized control plane with lightweight communication protocols boosts application-level message performance (T4.1).
- Containerized device workflows and OTA packaging reduce application deployment time and support diverse architectures (T5.1, T5.2).
- Containerized device workflows and OTA packaging reduce application deployment time and support diverse architectures (T5.1, T5.2).

Relevant EMPYREAN objectives include:

- F_ASSOC.1, F_ASSOC.5 – Enabling Association-aware orchestration and configuration.
- F_DI.1, F_DI.2 – Secure and flexible IoT device management.
- F_ST.3, F_ST.4 – Ensuring secure bootstrapping and operational trust.
- F_GR.1, NF_GR.2 – Support for zero-trust and scalable integration mechanisms.

5.3 Architecture

The cloud-native IoT management framework spans multiple architectural layers of EMPYREAN, aligning with the multi-layered approach described in the project's architecture (see Section 2). It integrates into:

- Device Layer: ESP32-class microcontrollers embed cryptographic UDS and generate EATs during onboarding.
- Edge Layer: K3s-based Kubernetes clusters run lightweight workloads, device discovery handlers, and OTA agents.
- Cloud Layer: Centralized orchestration logic (k8s operator) manages device lifecycle and OTA tasks.

- Application Layer: Web-based dashboards and APIs offer monitoring, OTA control, and device insights.

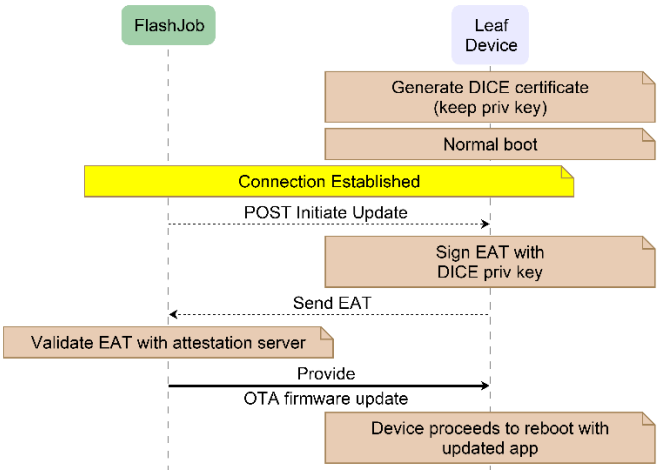


Figure 8: Far-Edge Device OTA Update process.

The system is composed of several microservices and operators:

- Device Onboarding Service (low-level attestation server): Handles EAT verification and device registration using a vendor-maintained board ledger.
- Akri Integration: Akri-based discovery handlers detect and register IoT devices as k8s resources.
- OTA Update Operator (integrated with Akri): Issues flashjob pods to deliver and apply firmware updates packaged as OCI images (Device-specific process shown in Figure 8, CRD and operator logical diagram shown in Figure 9).
- vAccel Offloading Agent (integrated as a multi-tier component): Enables ESP32 devices to forward AI/ML tasks to local edge accelerators.

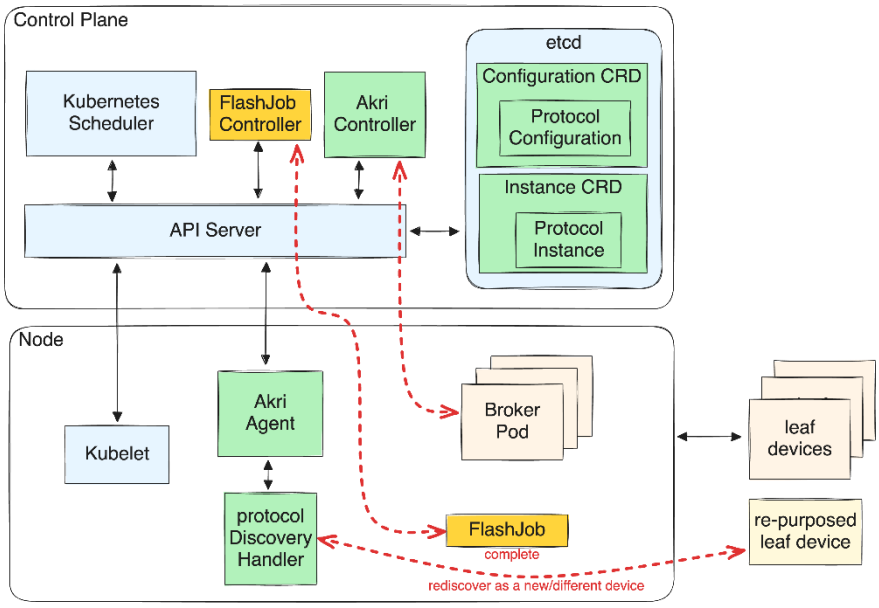


Figure 9: Operator-initiated FlashJob to re-purpose a Far-Edge device

5.4 Implementation

The cloud-native IoT management component is implemented using containerized microservices deployed within the EMPYREAN platform:

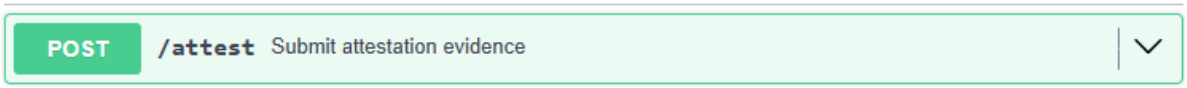
- Programming Languages: Go (operator/controller), C (device-side logic), Python (support services)
- K8s CRDs: Extend Akri's resource model for IoT-specific semantics (e.g., attestation state, firmware version, flashjob operator)
- Security Libraries: `mbedTLS` for device-level TLS, OpenSSL for attestation verification
- Flashjobs: Lightweight, stateless pods delivering firmware via HTTP(S) to target devices

Firmware images are delivered using OCI manifests, supporting full and delta updates for bandwidth efficiency.

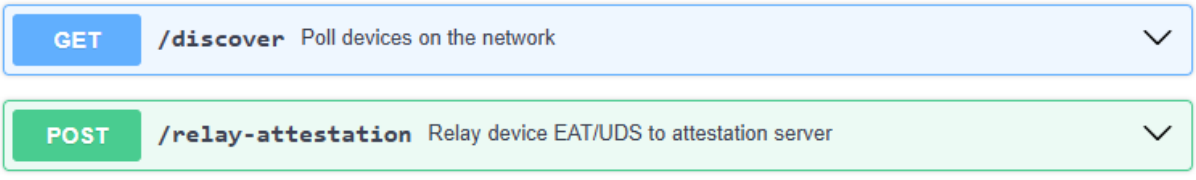
5.5 Public APIs

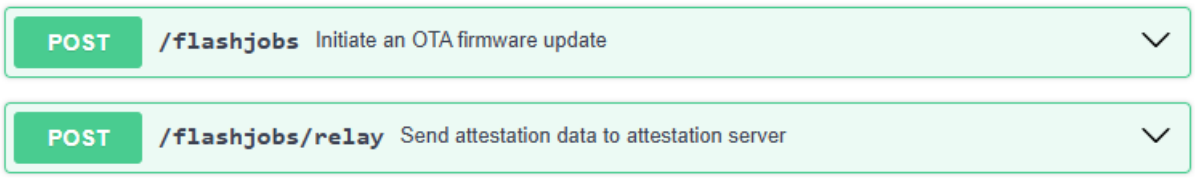
The NUBIS Component exposes several REST and K8s-native APIs:

Attestation server:

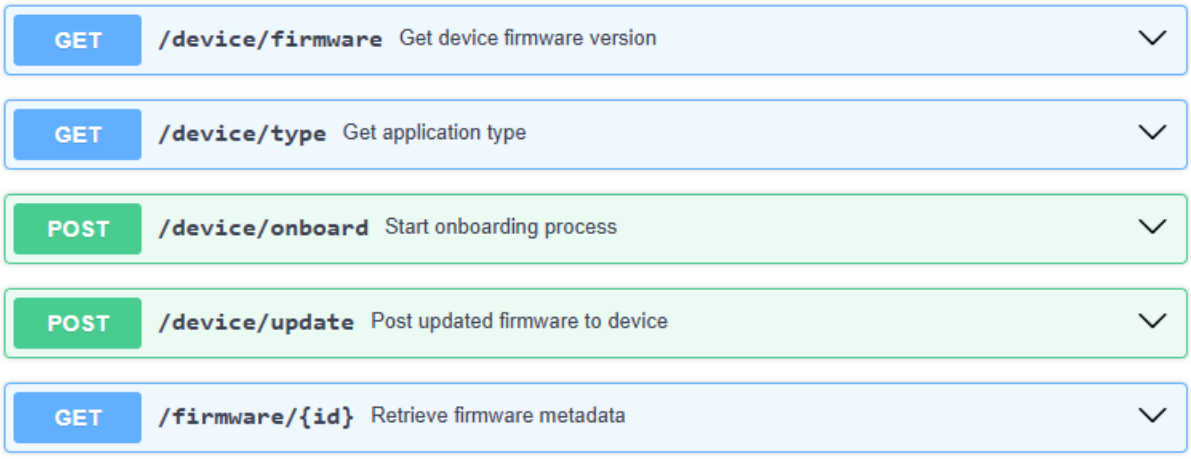


Discovery handler:





Device:



5.6 Integration with EMPYREAN Platform Services

The cloud-native IoT device management component integrates with several core EMPYREAN services:

- Privacy and Security Manager: Onboarding process and OTA policies integrate with PSM's credential verification and policy enforcement.
- Telemetry and Observability: Exposes device metrics and update states for monitoring and diagnostics.
- Edge Storage Service: Flashjobs may reference firmware stored in edge-accessible S3-compatible buckets.
- Workflow Engine: Can dynamically repurpose devices by triggering OTA updates through integrated workflows.

5.7 Relation to Use Cases

The cloud-native IoT device management component supports EMPYREAN's use cases by enabling secure, agile, and context-aware IoT deployments:

UC1: Secure Robotic Operations in Manufacturing

Ensures only attested robotic controllers are enrolled; allows safe OTA updates without human intervention.

UC2: Privacy-Preserving Agriculture Monitoring

Automates onboarding and firmware lifecycle of field-deployed sensors; supports edge inference via vAccel.

6 Edge Storage Service

6.1 Overview

The Edge Storage Service (ESS) provides a convenient, privacy-focused object store for association applications. It employs and manages both cloud and edge resources seamlessly, providing great flexibility in meeting application requirements. It achieves this by utilizing the technologies and some of the software components of SkyFlok, a file storage and sharing service developed by Chocolate Cloud.

As an object storage service, applications upload and download objects, organized into buckets. Compared to file system semantics, object stores provide a simplified concurrency model, great performance at scale, and are widely used in cloud storage whenever data is unstructured and of considerable volume. Beyond the public clouds, object storage solutions like Min.IO also exist for the edge (or private cloud). Commercially available hybrid solutions that can utilize both types of storage resources are uncommon.

To ensure high reliability and availability, most storage services employ replication, which provides great performance and simplicity. The downside is storage cost, especially when a high level of availability is desired (each replica adds an additional 100% storage cost compared to the original size). Erasure coding solves this issue by creating linear combinations of the original data and spreading them across storage locations. Redundancy is added to achieve high reliability as new linear combinations. This approach is much more cost-effective. For example, a configuration that allows any 2 out of 4 locations to reconstruct the original data has an overhead of 100%. A comparable 3-way replicated system requires 200% overhead. The difference is much greater if higher levels of availability are required as storage costs scale linearly for replication and sub-linearly for erasure coding.

6.2 Novel features

The Edge Storage Service uses a novel erasure coding technique called **Random Linear Network Coding (RLNC**), which has significant privacy benefits compared to traditional techniques such as Reed-Solomon. It is also a rateless code, meaning that redundancy can be added later, without the need to change previously stored data. RLNC is a patented technology, giving Chocolate Cloud a meaningful advantage in the secure cloud storage market.

Coded fragments are distributed across cloud and edge locations, achieving a **hybrid storage** solution that seamlessly combines the benefits of both types of resources. Users define a storage configuration using a **storage policy**. If better latency is desired, an edge-focused policy that stores most fragments using the Association's resources should be used. Conversely, a cloud-focused policy usually provides better reliability and availability at a lower

cost - ideal for long-term archival storage. This flexibility in working across the edge-cloud continuum is one of the distinguishing features, when compared to state-of-the-art commercial solutions.

To make it as simple to use as possible and require the least amount of learning and integration work from application developers, the service offers an **S3-compatible** API, featuring full coverage of bucket and object CRUD endpoints as well as more advanced features such as **multipart uploads** and **range queries**.

Many of these features have been tested and developed in SERRANO[9] (Horizon 2020 project) as part of a prototype system. For EMPYREAN, Chocolate Cloud has decided to bring its production-ready to the project, instead of continuing development of the prototype system. This choice was made to streamline the exploitability of EMPYREAN results and to start from a more mature software foundation.

EMPYREAN will bring two key, novel new features. The **temporary autonomous edge-only operation** makes it possible to keep serving read and write requests using Association-local requests when the link to the cloud is severed. Once reconnected, changes are synced, and normal operation can resume. The second feature significantly enhances the privacy characteristics of the system. By performing encryption and **storing encryption keys at the edge**, platform applications get further assurance that their data cannot be accessed by not only third parties, but also the storage provider itself. Additionally, several new S3 features will be developed as part of the project including **object versioning** and **access control,** improving coverage of the S3 API**.** All of these improvements have been selected with exploitability in mind.

6.3 Relation to project objectives and KPI

The Edge Storage Service will help in measuring the following project-level technical KPIs. T3.4 requires measurements to be performed, comparing edge-only storage policies to cloud-based ones. T3.5 concerns temporary autonomous operation when the link to the cloud is severed. This will be assessed using tests.

Table 3: EMPYREAN Technical KPIs related to the Edge Storage Service

ID	Indicator	Success Criteria	Objective
T3.4	Time reduction to read/write data when storing data purely on the edge compared to storage on the cloud.	by 40%	Obj.3
T3.5	Ability to access data stored on the edge when the link to the cloud is severed.	-	Obj.3

[9]SERRANO project website: https://ict-serrano.eu

The component is relevant in achieving several project objectives:

- F_ASSOC.1,F_ASSOC.4, F_ASSOC.5
- F_DI.4
- F_ST.1
- F_GR.5, F_GR.6
- F_DCM.1
- NF_GR.1, NF_GR.4, NF_GR.5, NF_GR.6

6.4 Architecture and Implementation

The overview of the architecture of the Edge Storage Service is shown in Figure 10. EMPYREAN user applications interact with the service through the **Edge Storage Gateway** (ESG), when inside the private network of their Association, or through the **Cloud Storage Gateway** (CSG). Both provide an S3-compatible API. The gateways perform the data processing tasks such as erasure coding and encryption, and they are in charge of distributing data to the storage locations. The CSG is only able to access **Cloud Storage** locations, whereas the ESG can store and retrieve fragments from Association-local **Edge Storage**. Much of the business logic is provided by the **SkyFlok.com** backend, which also manages different metadata. Among the many backend services, three are particularly relevant to EMPYREAN. The Edge Support Service is in charge of managing Association-related data, including ESGs and Edge Storage devices. The Migration Service is used to synchronize data after a network outage. The S3 API Service implements the S3 endpoints not directly linked to uploads and downloads.

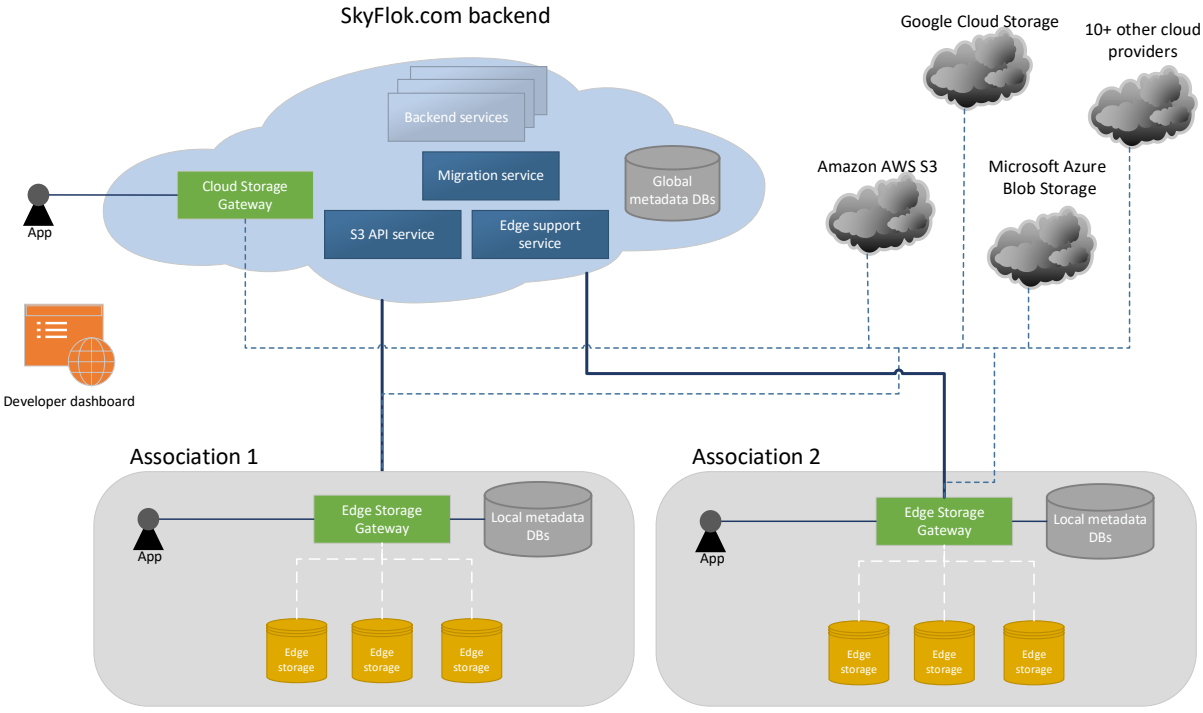


Figure 10: An overview of the main components of the EMPYREAN platform's Edge Storage Service.

Each Association has its own ESG. For privacy reasons, the Edge Storage Service provides no direct way to share or migrate data between Associations. Instead, applications that must work across Associations will need to use the EMPYREAN platform's security mechanisms to retrieve credentials for each Association separately.

6.4.1 Edge Storage Gateway

The Edge Storage Gateway (ESG) is the most important edge component of the service. It provides all APIs used by the EMPYREAN applications as well as the platform services.

The ESG is written in Python and uses a local relational database to store information needed for providing autonomous operation. The performance-critical erasure coding uses NCLib, a cpp header-only library developed in-house by Chocolate Cloud.

During normal operation, the ESG forwards S3 requests that do not involve data upload or download to the SkyFlok.com backend's S3 API service. Object uploads and downloads are done by the gateway, including all data processing such as compression, erasure coding and encryption.

The ESG is also a key component of the Analytics-friendly Distributed Storage, described in Deliverable 4.1 (M15).

6.4.2 Cloud Storage Gateway

The Cloud Storage Gateway (CSG) has a shared codebase with Edge Storage Gateway and is a core component of SkyFlok S3, Chocolate Cloud's upcoming object storage solution. However, it lacks all features related to associations. The joint development of the two gateways has its benefits when it comes to writing validated, maintainable code. It has the added benefit that S3-related features added to either component are either automatically inherited by the other or can easily be migrated.

6.4.3 Cloud storage

SkyFlok, CC's main commercial product, stores coded fragments in cloud-based object storage. As such, the Skyflok.com backend has connectors to both the proprietary interfaces of the major cloud providers (e.g., Google Cloud Platform, Azure) and the interfaces that are used by several providers like AWS's S3 and OpenStack Swift's native API. Thanks to this integration, the Edge Storage Service is able to distribute data across the globe to more than 110 cloud locations, as illustrated in Figure 11. The list is constantly being expanded.

Figure 11: Map showing supported cloud storage locations across the globe. Up to date as of March 2025.

The Gateway uploads and downloads fragments to and from cloud locations directly, using pre-signed URLs that are supplied by the SkyFlok.com backend. Each URL contains a time limit in which it can be used and has a precisely defined scope in terms of what it can be used. To ensure its veracity and that has not been modified, the backend cryptographically signs it, hence its name.

6.4.4 Edge Storage

Edge Storage devices are practical representations of individual Association storage resources. It is a layer of abstraction that provides two key benefits. Firstly, integration into the storage system is greatly simplified by wrapping a storage resource within a software component. Wrapping each storage resource type within the same component ensures a common API. The system does not need to have special considerations for the characteristics of the underlying resource. Secondly, selecting a technology that can wrap seamlessly around a large variety of storage resources makes Edge Storage devices a flexible, user-friendly mechanism.

To achieve these goals, we have selected Min.IO, a self-hosted object storage solution. It exposes an S3-compatible API, making integration with the Gateway simple, given S3's popularity in the cloud object storage space. Another benefit of Min.IO is its ease of use in a containerized environment. By deploying it to a K8s cluster or even running it as a standalone Docker image, it can store all data and metadata on a storage volume mounted on the container. This opens up a wide range of storage types that can be seamlessly used, such as host file systems, iSCSI, NFS, and more. Furthermore, Min.IO uses the Prometheus Data Model to provide support for monitoring and alerting. This widely used standard API and data model makes it possible for the Telemetry service to monitor the Edge Storage devices.

6.4.5 SkyFlok.com backend

SkyFlok is a next-generation file sharing and storage solution for users who care deeply about privacy and security. It is a multi-cloud platform which distributes data across a wide range of

commercially available clouds. Beyond the big three of Amazon, Google and Microsoft, SkyFlok supports most major EU cloud providers and can be configured to be GDPR compliant (66 out of total of 111 cloud locations are GDPR-compliant). A key enabler of this is the ability provided to users to select the cloud providers that will store their data as well as the actual locations down to the city level. Internally, SkyFlok's secret sauce is RLNC, an erasure code that provides reliable service even if a cloud provider becomes unavailable. It also offers protection from data loss and gives privacy benefits beyond those provided by conventional encryption.

SkyFlok was launched in February 2018. Since then, over 750 SME teams have used the service. Chocolate Cloud launched in 2020 its reseller portal for resellers in multiple countries (incl. Canada, Italy, Denmark, UK) to commercialise SkyFlok.

The Edge Storage Service relies on the software infrastructure behind SkyFlok, the Skyflok.com backend, for a wide range of features. These can be grouped as follows:

- Object storage metadata management
- Storage location management
- Generating pre-signed upload and download links
- Storage policy management
- File and metadata consistency checking
- Authentication and authorization
- User and team management

The SkyFlok.com backend is implemented as 15+ Python microservices. Each manages its own business entities through either a MySQL database or Google Datastore, a NoSQL database offered by the Google Cloud Platform.

Among the microservices, two are particularly relevant for the Edge Storage Service. The **S3 API Service** performs all S3 operations unrelated to uploading or downloading objects. The cloud and edge gateways communicate with it directly, forwarding requests with minimal changes. For Association and edge storage-related features, CC has implemented the **Edge Support Service** for the project. It manages all data related to associations, ESGs, and Edge Storage devices.

6.4.6 Developer dashboard

The Developer dashboard provides a simple web-based user interface to application developers. It provides an overview of S3 buckets and objects and, through either the cloud or edge storage gateways, makes it possible to upload and download objects and manage buckets. S3 API keys can also be defined, providing fine-grained access control to data.

Beyond the S3 features, the dashboard makes it possible to define and manage storage policies. Users can specify exactly where their data should be distributed using which

redundancy scheme, and they can also select between supported compression and encryption methods.

Finally, the dashboard provides management of EMPYREAN's storage-related entities. This includes a way to manage Associations, Edge Storage devices, and Edge Storage Gateways. Each Association owner must create their association using the dashboard, defining at least one gateway. Non-owner teams can join associations using a secret key that they obtain from the owner. Figure 12 shows a list of Associations the currently logged in user is a member of.

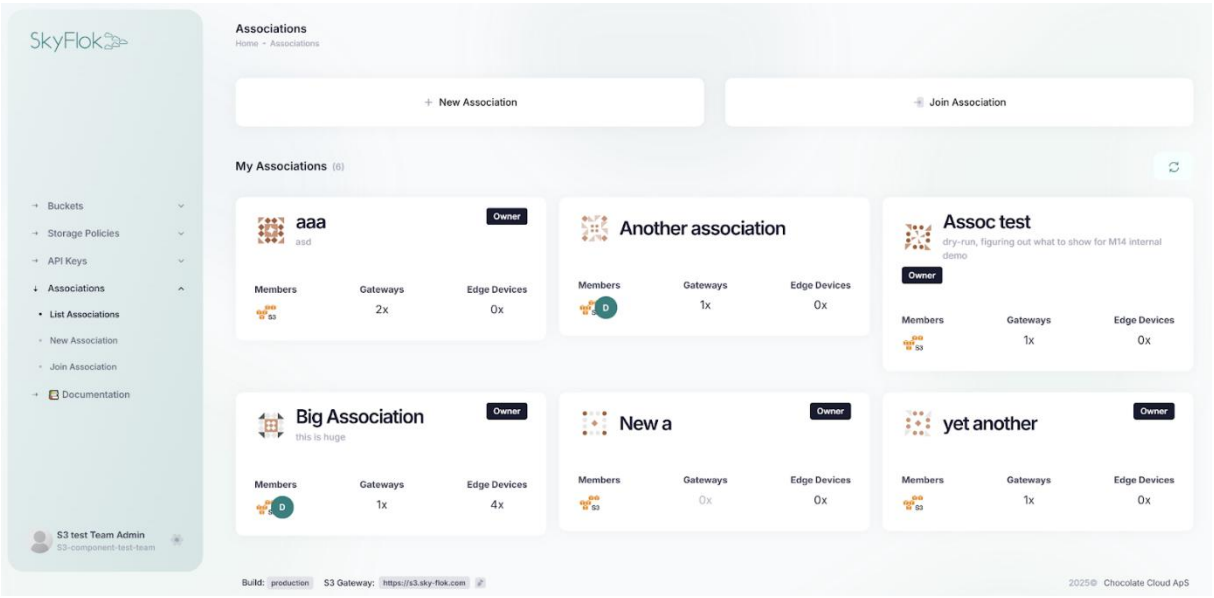


Figure 12: The page in the Developer dashboard that lists associations the user is a member of.

6.4.7 Deployment

The service's components are deployed at various points of the edge-cloud continuum. The Edge Storage Gateway and Edge Storage run inside an Association, ideally in a managed K8s cluster. They can be deployed either directly using YAML descriptors or through a helm chart. The Edge Storage is unique in that it relies on the K8s StatefulSet mechanism for simple deployment. Both components can also be deployed directly as Docker containers or can even be adapted to run on bare metal.

The Cloud Storage Gateway is deployed to fly.io's extensive global infrastructure. This employs some of the same concepts of a Content Delivery Network in that requests are automatically routed using DNS to the nearest available gateway. The CSG is managed by Chocolate Cloud and is not tied to any single Association. The SkyFlok.com backend is deployed to the Google Cloud Platform as microservices running in Google App Engine Standard.

6.4.8 Workflows for temporary autonomous operation

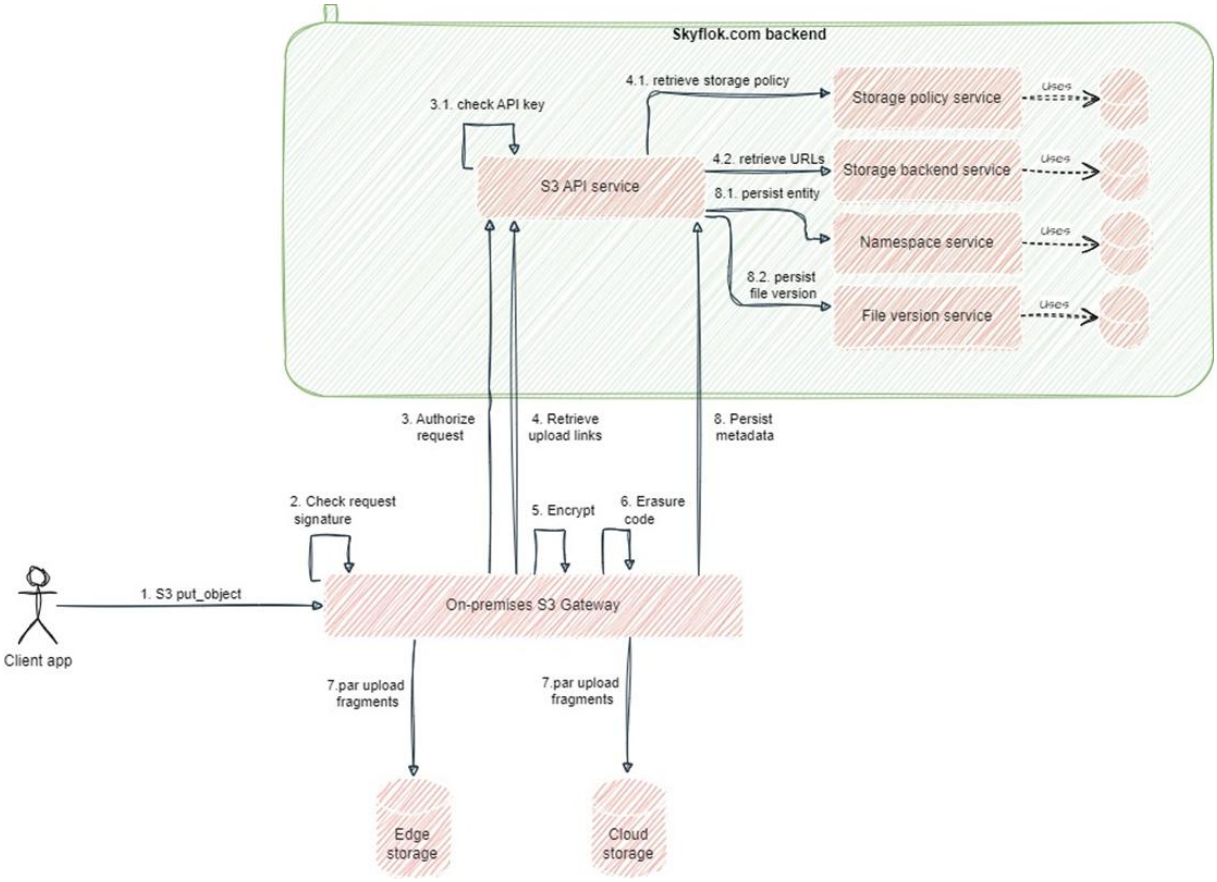


Figure 13: Communication diagram showing the upload workflow when the Association's Gateway is connected to the outside world, including the cloud-based SkyFlok.com backend.

In this subsection, we present the workflows related to file upload, namely the S3 PutObject endpoint, using the Edge Storage Gateway. Figure 13 shows regular operation where the Association has internet connectivity and both the SkyFlok.com backend and cloud storage locations are accessible. The Gateway performs the verification of the request, checking that it has not been tampered with, but delegates authentication and authorization to the backend. This provides the pre-signed URLs, based on the storage policy which the ESG uses to upload fragments. This is done, after encryption and erasure coding take place. The result is then recorded on the backend.

In the event the Association's outside connection is not available, the Edge Storage Service can continue to function in a degraded capacity. Figure 14 shows how file uploads are handled. The Gateway must temporarily take on some of the backend responsibilities by authorizing requests, generating pre-signed URLs, and recording results. To achieve this, a colocated metadata database stores the required business entities. Given that no cloud locations are reachable in this capacity, fragments can only be uploaded to Edge Storage devices.

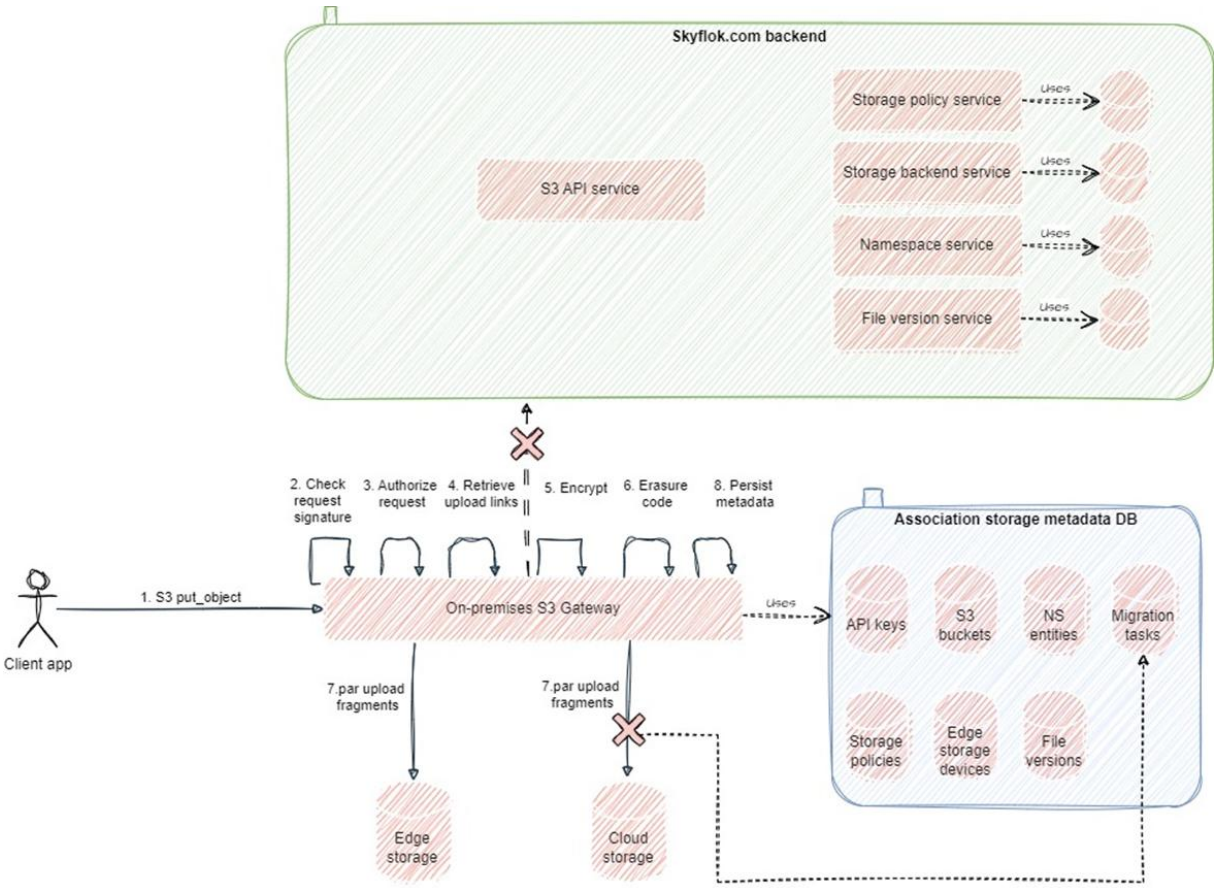


Figure 14: Communication diagram showing the alternative upload workflow when the Association's Gateway is temporarily disconnected from the outside world and thus must itself assume some of the roles of the cloud-deployed SkyFlok.com backend.

6.5 Public APIs

Object storage (S3) API

The ESG's S3-compatible object storage API supports all major Create, Read, Update, Delete (CRUD) features of both objects and buckets in their simplest form.

Buckets

- CreateBucket
- DeleteBucket
- ListBuckets

Objects

- GetObject
- HeadObject
- PutObject
- ListObjectsV2
- DeleteObject

An API reference can be found on Amazon's website[10]. Amazon Web Services S3 provides two URL schemas to access buckets and their contents. Like most Secure Storage API adopts the first one. The second scheme was created by Amazon to address the bottlenecks related to routing requests through DNS on a global scale. Given that the Edge Storage Service operates at a much more modest scale, the second schema does not bring any benefits.

1. *http://s3.amazonaws.com/[bucket_name]/*
2. *http://[bucket_name].s3.amazonaws.com/*

The Secure Storage API uses the same parameters for each endpoint and maintains the error handling of AWS S3, both in terms of the format of error messages as well as the different codes that identify the cause.

Storage Policy API

The Storage Policy API allows EMPYREAN users to programmatically create and retrieve storage policies. These can be thought of as recipes used to translate an application's storage requirements into a storage resource allocation.

Storage Resource Telemetry API

The list of supported Cloud Storage locations is provided by the ESG, listing their static characteristics as well. Beyond this, the performance of cloud locations is continuously monitored by the SkyFlok.com backend. This information is also exposed by the ESG through its Telemetry API.

The list of Edge Storage devices associated with an association is provided by the ESG, including their static characteristics. Dynamic characteristics can be collected directly through a Prometheus-compatible interface exposed by MinIO. No communication with the ESG is needed. This interface provides a way to both monitor performance characteristics as well as configure alert rules for certain types of abnormal events.

6.6 Integration with EMPYREAN Platform services

The Edge Support Service will be an integrated part of the EMPYREAN platform, achieving its purpose through connections to other services. As such, it will utilize many of the components needed to manage associations. In the following, we constrain ourselves to presenting the integrations that are specific to the storage-related aspects.

The Privacy and Security Manager (UMU), provides decentralized identity management, and secure authentication for platform applications. Through its policy engine, usage quotas can be placed on storage resources as smart contracts and then enforced by the ESS. The PSM, by

[10] S3 API reference: https://docs.aws.amazon.com/AmazonS3/latest/API/API_Operations_Amazon_Simple_Storage_Service.html

working across Associations, also opens up the possibility for using storage in applications that work across Association boundaries.

User applications define data workflows using the Ryax Workflow Manager (RYAX). One key functionality is the ability to dynamically allocate storage resources tailored to the specific requirements of each workflow. Each S3 bucket in Edge Storage is governed by a storage policy, enabling Ryax workflows to seamlessly direct data to/from edge resources for low-latency access, cloud storage for scalability and reliability, or a combination of both through the use of hybrid storage policies.

Applications leveraging the Decentralized and Distributed Data Manager (ZSCALE) can easily utilize the services of the ESS. Eclipse Zenoh, and thus Zenoh-Flow, has an S3 plugin, meaning applications have a simple way of including the service in their data pipelines.

Finally, the Telemetry Service (ICCS) monitors Edge Storage devices as well as cloud locations through information provided by the Edge Storage Gateway and the devices themselves.

6.7 Relation to use cases

The Edge Storage Services provides a convenient way to store data for EMPYREAN applications. This includes the two use case applications:

UC1: Secure Robotic Operations in Manufacturing

There are several types of data that could be stored by UC1, including machine learning models, datasets to train the models as well as inference results. Object storage is especially useful if multiple versions of models need to be kept for a longer period of time. To account for the stringent privacy requirements of the use case, edge-only storage policies can be used to ensure that data is never stored outside IDEKO's premises.

UC2: Privacy-Preserving Agriculture Monitoring

UC2 works with large volumes of remote and proximal sensing satellite data, requiring long-term storage. This is an ideal scenario for showcasing the benefits of object storage. Furthermore, like UC1, machine learning models can also be stored using the Edge Storage Service. UC2 performs computations across the cloud continuum, meaning that some data may best be kept either in the cloud, or using a hybrid policy in both the cloud and at the edge. To enhance privacy for cloud-based policies, file encryption keys can be stored at the edge, further limiting the potential for improper access.

The feedback gathered from the use case providers will be used to improve the service in the following phase of the project. Furthermore, it will be essential in guiding Chocolate Cloud's (CC) individual exploitation plans. By highlighting the most relevant challenges and most market-ready features, CC aims to establish which parts are to be added to its current products.

7 Decentralized and Distributed Data Manager

7.1 Overview

EMPYREAN uses Eclipse Zenoh as its decentralized and distributed data framework. Zenoh is a Pub/Sub/Query protocol designed to provide a set of unified abstractions for dealing with data in motion, data at rest, and computations at Internet scale. Eclipse zenoh technology was initially introduced in D2.1[11], and later refined in D2.3[12].

One of the main characteristics of Zenoh is openness and interoperability. This is aligned with EMPYREAN's openness objectives, enabling diverse technologies to collaborate. For example, the new Zenoh's querier API supports efficient and optimized data retrieval. Enhanced support for ROS2[13], a widely used framework in robotics, strengthens connections between Zenoh and other platforms, facilitating interoperability.

Figure 15 presents the types of different plugins that Zenoh supports at the moment. As storage plugins, it supports InfluxDB[14] storage plugin for time series databases, RocksDB[15] for key-value oriented databases, Amazon S3 to store data as objects in buckets in the cloud, and MinIO for high-performance, distributed object storage system.

Runtime plugins include Zenoh-Flow[16], which is a dataflow programming framework that enables declaring an application as a set of computations that can be distributed in a set of nodes across the continuum. For the protocols plugin, Eclipse Zenoh supports bridges with the Data Distribution Service (DDS) standard, with the REST-based web APIs, and with MQTT[17]-based protocols. Additionally, Eclipse Zenoh has been selected as an alternative communication middleware for the ROS2 release, and a rmw_zenoh is under development to become a Tier-1 protocol for the Robot Operation System (ROS) community.

7.1.1 Current Developments

In October 2024, Eclipse Zenoh v1.0.0 was released, marking a milestone for the open-source project as it came out of the incubation period to become a fully Eclipse project. That has not prevented it from moving forward. Just a couple of months later, by the time of writing this deliverable, March 2025, the latest release is Zenoh v1.3.0. This continuous update reflects the engagement with the requirements and priorities, such as adaptability, security, and

[11] Deliverable 2.2: Initial Release of EMPYREAN Architecture. September 2024.
[12] Deliverable 2.3: Final EMPYREAN architecture, use cases analysis and KPIs. January 2025.
[13] ROS2: https://docs.ros.org/en/foxy/index.html
[14] InfluxDB: https://www.influxdata.com/
[15] RocksDB: https://rocksdb.org/
[16] Baldoni, G., Loudet, J., Guimarães, C., Nair, S. and Corsaro, A., 2023, October. A Data Flow Programming Framework for 6G-Enabled Internet of Things Applications. In 2023 IEEE 9th World Forum on Internet of Things (WF-IoT) (pp. 1-8). IEEE. https://ieeexplore.ieee.org/abstract/document/10539539
[17] MQTT: https://mqtt.org/

performance optimization, ensuring an efficient and interconnected ecosystem for devices, applications, and data. Other important characteristics include:

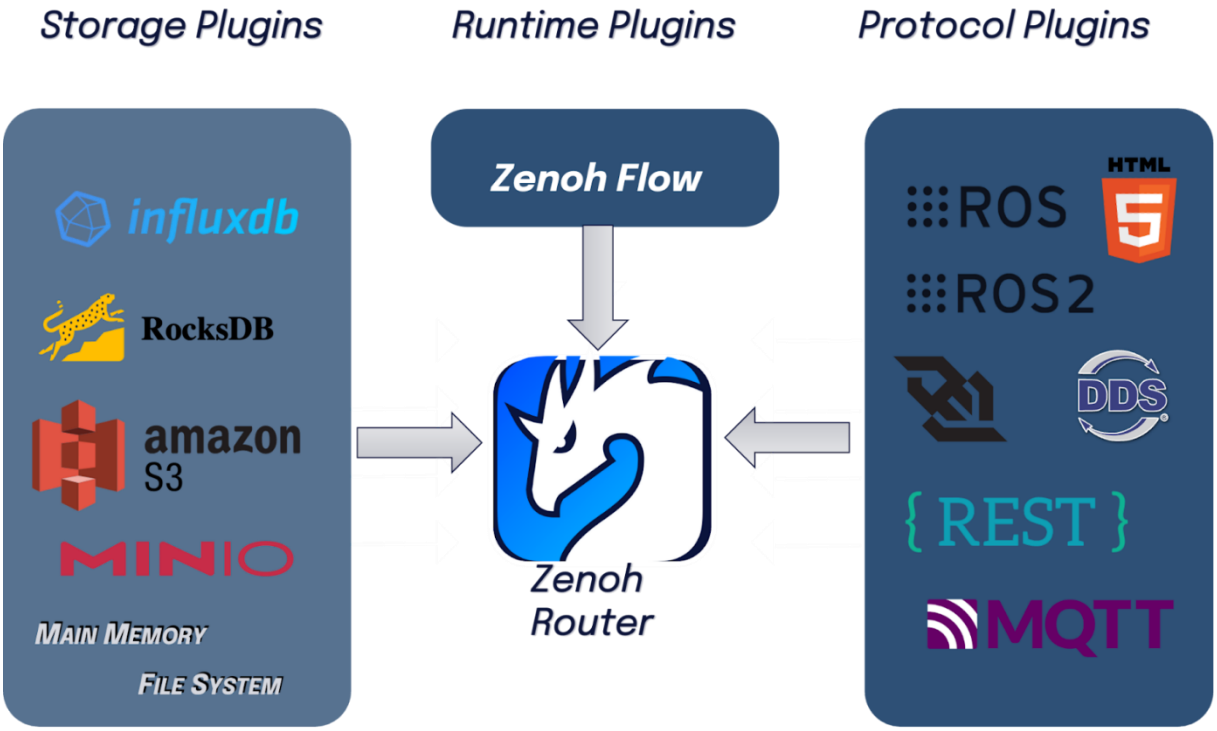


Figure 15: Eclipse Zenoh plugins.

Adaptability and Scalability: new features that adapt to varying technological needs. The stabilization of liveliness API ensures real-time monitoring of active participants in the network. Zenoh-Pico, which is made for small IoT devices, has also been updated. With huge improvements in performance and extension in scope, as it is now compatible with the Raspberry Pi Pico series.

Security and Privacy: The protocol enhancements address critical issues such as fragmentation and message integrity, ensuring secure and reliable data transmission. These updates safeguard interactions across the network, whether between IoT devices or in the cloud. Zenoh supports Transport Layer Security (TLS) as a transport protocol. TLS can be configured in two ways:

- Server side authentication: clients validate the server TLS certificate but not the other way around, that is, the same way of operating on the web where the web browsers validate the identity of the server via means of the TLS certificate.
- Mutual authentication (mTLS): where both server-side and client-side authentication is required. The configuration of TLS certificates is done via a configuration file.

Technology Agnosticism: to be considered a part of the EMPYREAN's platform, Zenoh should support diverse types of hardware infrastructure and software platforms, including new features for QNX operating systems[18], highlighting its technology-agnostic approach. This openness reduces dependence on specific vendors and encourages widespread adoption.

7.1.2 Beyond the state-of-the-art

Existing protocols were designed to work on a very specific use case and address a connectivity island. As an example, the Data Distribution Service (DDS)[19] was designed to provide a pub/sub protocol that works best for applications running on resourceful hardware connected by multicast-enabled (UDP/IP) wired Local Area Network (LAN). Another assumption in DDS's design is that peer-to-peer communication is quintessential and most of the applications consume data from every other application.

On the other hand, there is MQTT, which was designed to support pub/sub via a client-to-broker architecture over TCP/IP networks. What is interesting is that both DDS and MQTT provide pub/sub. Yet, their implementations force onto the user very specific communication topologies that are completely orthogonal to the concept of pub/sub. This introduces architectural inflexibility and scalability issues. As an example, DDS is notoriously hard to work with and scale on a Wide Area Network (WAN) as a consequence of its (flat) peer-to-peer only model and its reliance on multicast IP. MQTT, makes communicating across a WAN easy, as far as the user is willing to accept having a hub-and-spoke architecture and a topology not ideal for several edge applications. Therefore, the solution has been to use different protocols on different segments of the system and integrate them, hoping to have some meaningful end-to-end semantics. This is tedious, error-prone, and inefficient. This is a consequence of the inability of established protocols to deal with the cloud-to-device continuum.

Zenoh is a Pub/Sub/Query protocol that provides a set of unified abstractions to deal with data in motion, data at rest, and computations at large scale. Zenoh runs efficiently on server-grade hardware and networks as well as on microcontroller and constrained networks.

7.2 Architecture and implementation

Zenoh operates over resources. A resource is a (key, value) tuple, where the key is an array of arrays of characters. When representing keys, we usually use the "/" as a separator. Thus, ***home/kitchen/sensor/C2O2*** is a resource key.

[18] QNX OS Architecture and Concepts: https://www.qnx.com/developers/docs/8.0/com.qnx.doc.neutrino.prog/topic/overview.html
[19] Data Distribution Service (DDS). OMG 2018. https://www.omg.org/spec/DDS/1.4/About-DDS

A set of keys can be expressed by means of a key selector, which may include "*" or "**" that expand respectively to an arbitrary array of characters not including the separator, and an array of arrays of characters. For instance, ***home/kitchen/sensor/**** would represent the set of keys, including all the sensors in the kitchen, while ***home/*/sensor/C2O2*** would represent the set of keys representing all the C2O2 sensors in the house.

Zenoh allows the selection of a set of resources by using a selector. The syntax supported Zenoh's selector is ***keyexpr?arg1=val1&arg2=value*** – where ***keyexpr*** is a key expression as defined above. Some arguments, such as those for indicating filters, projections, and time intervals, are built-in; application-specific semantics can be added by defining additional arguments. As an example, the selector ***home/*/sensor/temperature?_filter="temp>25"&_project="hum"***, among all the temperature sensors in the house, it would select those whose value is greater than 25 and project their humidity.

Moreover, the Zenoh protocol defines three different kinds of network entities: (i) publishers, (ii) subscribers, and (iii) queryables. A publisher should be thought of as the source for resources matching key expressions. As an example, a publisher could be defined for a key, such as ***home/kitchen/sensor/C2O2***, or a set of keys, such as, ***home/kitchen/sensor/**** or ***home/kitchen/*****.

Symmetrically, a subscriber should be thought of as a sink for resources matching key expressions. As an example, a subscriber could be defined for a key, such as ***home/kitchen/sensor/C2O2***, or for a set of keys, such as, ***home/**/sensor/****.

A queryable should be through of as a well for resources whose key match a key expression. As such a queryable for ***home/kitchen/**** essentially promises that if queried for keys that match this key expression it will have something to say.

7.3 Public APIs

The main core of the zenoh protocol implementation is done in Rust. Additionally, there are APIs in different programming languages, such as:

- Rust, https://docs.rs/zenoh/latest/zenoh/
- C/C++, https://github.com/eclipse-zenoh/zenoh-cpp
- Kotlin (Java), https://github.com/eclipse-zenoh/zenoh-kotlin
- Python, and https://github.com/eclipse-zenoh/zenoh-python
- Typescript/Javascript https://github.com/eclipse-zenoh/zenoh-ts

7.4 Relation to use cases

Zenoh offers several orthogonal primitives that can be used by applications to declare a data exchange mechanism; these are:

Declarations: Namely, *declare_resource*, *declare_publisher*, *declare_subscriber*, and *declare_queryable* allow the declaring of a resource, a publisher a subscriber and a queryable respectively. A declaration is either used to optimize certain aspects of the protocol, such as automatically mapping keys to small integers, or to inform the rest of the Zenoh network that a specific endpoint is available.

Producing Data (Publisher). The *put* operation is used to produce a (key, value). This operation provides options that allow specifying the congestion control applied to it, the associated priority and a few other non-functional properties.

Subscribing Data (Subscriber). The *sub* operation is used to subscribe to a key-expression. This operation provides options that allow specifying the wildcards (i.e, *, **) to query for multiple key-expressions and also allows filter parameters.

Query. Zenoh provides a *get* operation that allows the issuing of a query. This query will be served by a set of queryable that cover, in a set-theoretical sense, the key expression portion of the query. Additionally, among all the set of sets that cover the query, Zenoh will select the one that is closest in routing terms. Zenoh provides options to control if only one of such set will be triggered or if all the matching queryable will. It also allows to control wether a partial cover is acceptable or not. The get operation also allows to control how data will be consolidated on the way back, and if consolidation is required at all.

Deleting Data. Zenoh provides a delete operation that makes it possible to indicate the desire that a resource shall be deleted.

7.5 Integration with EMPYREAN Platform services

There are two potential integration points of Zscale's Eclipse Zenoh with Ryax's Intelligent workflow orchestrator.

- A potential collaboration is to use Eclipse Zenoh as a generic Remote Procedure Call (RPC) to interconnect different Ryax intelligent workflow orchestrator's actions.
- Other potential integration point is through the use of cloud storages, Eclipse zenoh supports object storage either in the Amazon S3 buckets or MinIO storages through the Zenoh-backend-s3 plugin.

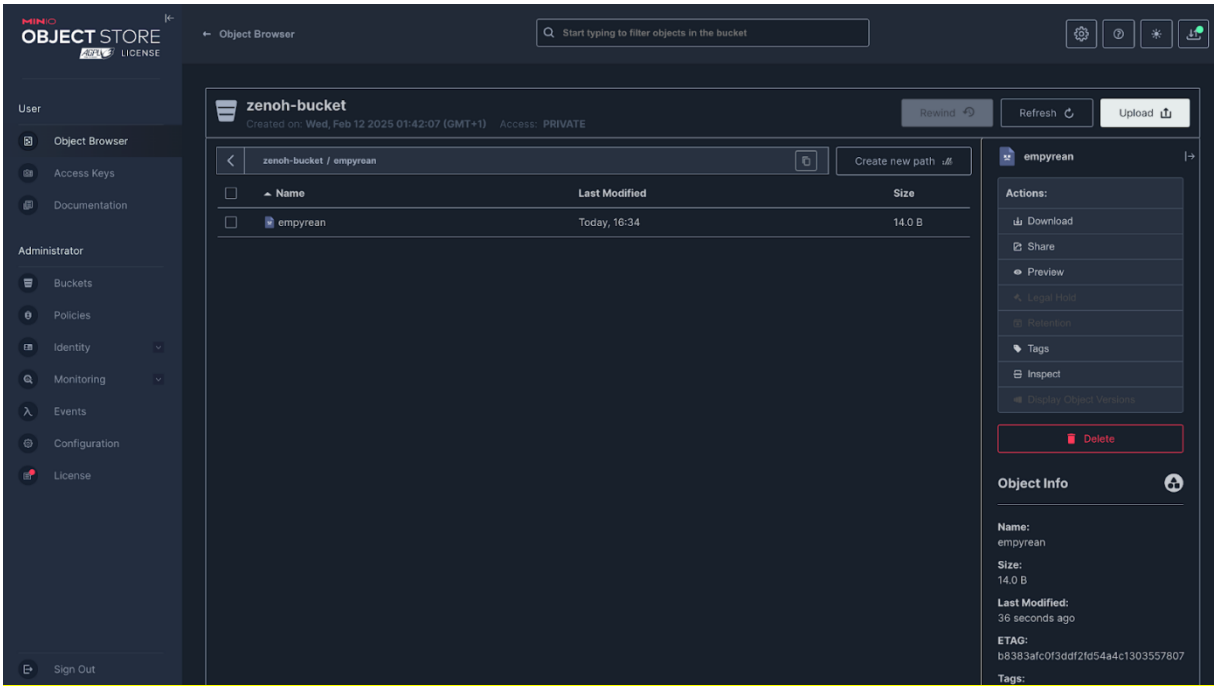


Figure 16: A MinIO object storage created with Eclipse zenoh backed plugin.

In Zenoh, a backend is a storage technology (i.e., DBMS, time-series database, file system., memory backend) allowing to store the keys/values publications made via Zenoh and return them on queries. Zenoh's backend relies on Amazon S3 to implement the storage. It is also compatible to work with MinIO object storage. Figure 16 shows a MinIO object storage created through Eclipse zenoh. Figure 17 represents the object detailed description.

Moreover, Zenoh is utilized as the core communication middleware to enable the new multi-agent capabilities within the EMPYREAN Decision Engine. Leveraging its lightweight, high-performance publish/subscribe and query-based communication model, Zenoh facilitates scalable, low-latency data exchange among distributed Decision Engine instances. Each instance, acting as an autonomous agent, uses Zenoh topics to coordinate actions, share telemetry insights, and negotiate workload distribution strategies across Associations. Deliverable D4.2 (M15) provides more details for this specific integration.

Finally, there is ongoing work to integrate the decentralised and distributed capabilities of Zenoh into EMPYREAN proposed use cases. In D2.3, TRACTONOMY presented their Advanced Inference and Coordinated Behaviours for Warehouse Automation Robots (UC3), and its integration with the Zenoh protocol for data communication in section 6.3. However, as it is known, unfortunately TRAC partner is no longer taking part in the project. At the time of writing this deliverable there are ongoing discussion about a potential UC3 replacement.

.

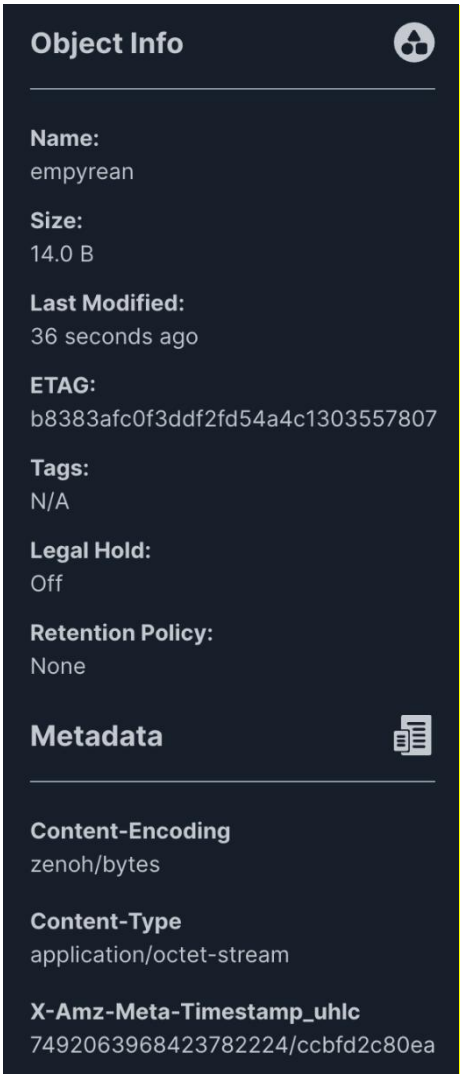


Figure 17: A MinIO object's detailed description.

8 Conclusions

The deliverable outlined how the technical outcomes of T3.1 and T3.2 play a key role in providing a robust security and trust model for the platform's applications. It described the technical details on how each component is being implemented including critical design decisions, internal architecture, and public APIs. The role of each component was also presented through its connection to the project's requirements, KPIs, and use cases.

To achieve the goals of Work Package 3 and, in general, the project, these components implement a wide range of features, many of them directly exposed to EMPYREAN application owners. Without any claim to completeness, this deliverable presented mechanisms for:

- providing identity management using decentralized identifiers,
- dynamic policy enforcement using blockchain-based smart contracts,
- a novel cryptographic library for privacy-preserving attribute-based credentials that can be used in a distributed environment,
- a cloud-native framework for providing secure code execution,
- a secure, distributed object storage system that works across the edge-cloud continuum and employs a novel privacy-preserving erasure coding scheme,
- a scalable, secure decentralized and distributed solution for managing data-centric workflows.

In the second phase of T3.1 and T3.2, the focus will fall on completing the integrations of these components to provide a coherent platform view to EMPYREAN application owners, including the project's use cases. Beyond this, each component will play a key part in validating that the project's technical objectives have been met and KPI target values reached. As such, WP3 results will continue to play an important role in the project's life cycle, being tied directly to efforts in WP5 and WP6.