HORIZON EUROPE FRAMEWORK PROGRAMME

NEARDATA

(grant agreement No 101092644)

Extreme Near-Data Processing Platform

D4.2 Data Broker Reference Implementation

Due date of deliverable: 31-10-2025
Actual submission date: 31-10-2025

Start date of project: 01-01-2023 Duration: 36 months

Summary of the document

Document Type	Report
Dissemination level	Public
State	v1.0
Number of pages	91
WP/Task related to this document	WP4 / T4.1/T4.2/T4.3/T4.4/T4.5
WP/Task responsible	TUD
Leader	André Martin (TUD)
Technical Manager	Pubudu Jayasena (TUD)
Quality Manager	Aaron Call (BSC)
Author(s)	André Martin (TUD), Pubudu Jayasena (TUD),Tran Ngoc Nhat Huyen(TUD), André Miguel (SCO), Alan Cueva (DELL), Raúl Gracia (DELL), Sebastian Bodenstedt (NCT), Max Kirchner (NCT)
Partner(s) Contributing	TUD, NCT, SCO, DELL
Document ID	NEARDATA_D4.2_Public.pdf
Abstract	This deliverable includes the final design and reference implementation for all tasks. It will also show final validations and the verification of goals for this WP.
Keywords	Data Broker, Control Plane, Confidential Compute, TEEs, Confidential Compute Layer, Confidential Orchestration, Federated Learning

History of changes

Version	Date	Author	Summary of changes
0.1	30-09-2025	Pubudu Jayasena (TUD),André Martin (TUD),André Miguel (SCO)	First complete draft.
0.2	17-10-2025	Pubudu Jayasena (TUD),André Martin (TUD),André Miguel (SCO)	Review feedback.
1.0	31-10-2025	Pubudu Jayasena (TUD),André Martin (TUD),André Miguel (SCO)	Final version.

Table of Contents

List of Abbreviations and Acronyms

AES-GCM	Advanced Encryption Standard with Galois Counter Mode
AI	Artificial Intelligence
API	Application Programming Interface
CAS	Configuration and Attestation Service
CRD	Custom Resource Definitions
CVE	Common Vulnerabilities and Exposures
DAG	Directed Acyclic Graph
EPC	Enclave Page Cache
FL	Federeted Learning
FSPS	File System Protection Shield
HPC	High-Performance Computing
IAM	Identity and Access Manager
IAS	Intel Attestation Service
IoT	Internet of Things
NPS	Network Protection Shield
POC	Proof Of Concept
SDK	Software Development Kit
SGX	Software Guard eXtensions
TCB	Trusted Computing Base
TCP	Transmission Control Protocol
TLS	Transport Layer Security
VM	Virtual Machine

1 Executive summary

The NEARDATA Data Broker, as presented in this final release and documentation, is a central element of the NEARDATA architecture (refer to D2.1) that orchestrates secure and trustworthy data flows across the Cloud/Edge continuum. Its primary objective is to provide robust data governance, enabling stakeholders to securely access, share, and transfer confidential data from distributed sources, while ensuring the confidentiality and integrity of both applications and their components. The Data Broker manages user access policies and roles, supports multiple confidential data access methods, and integrates seamlessly with existing NEARDATA frameworks and tools.

To demonstrate its capabilities in controlling data access and enforcing security policies, the Data Broker has been integrated with a federated learning application using the FLOWER framework and Trusted Execution Environments (TEEs) This federated learning example is referenced throughout the document to illustrate the practical enforcement of confidentiality and integrity within NEARDATA. Beyond federated learning, the Data Broker extends its confidential computing capabilities to metabolomics workflows by supporting confidential containers for both Kubernetes and Singularity compute backends, as well as for the MinIO storage backend. This ensures that all data processed by Lithops whether at rest in MinIO, in transit between services, or in use during computation, is encrypted and protected within secure enclaves. SCONE's policy-driven access control and attestation mechanisms guarantee that only authorized, attested services can access sensitive metabolomics data, delivering comprehensive confidentiality and integrity across the entire workflow.

A comprehensive evaluation has confirmed that the Data Broker achieves the following key performance indicators (KPIs): validated confidentiality and integrity; negligible performance overhead; and secure multi-user access with robust policy enforcement. These results are supported by quantitative benchmarks and practical demonstrations in both metabolomics and surgomics use cases, which confirm the effectiveness, scalability and suitability of the Data Broker for high-throughput, distributed workloads.

The document also details the suite of security tools developed to support these capabilities, including the SCONE runtime (enabling applications to run in TEEs), a cross-compiler, and the Sconify tool, which together allow developers to convert native applications into confidential ones and streamline secure deployment. By integrating federated learning (via FLOWER), secure streaming (via Pravega), and confidential computing (via SCONE), the Data Broker enables privacy preserving, scalable, and trustworthy data processing for modern healthcare and life sciences research.

In conclusion, this deliverable summarizes the Data Broker's achievements and outlines the research activities that have supported these advancements, positioning the Data Broker as a central enabler of secure, compliant, and future-proof data ecosystems within NEARDATA. The comprehensive approach ensures that sensitive data remains confidential and secure throughout its lifecycle, meeting the stringent requirements of contemporary scientific environments.

2 Introduction

The NEARDATA project, titled "Extreme Near-Data Processing Platform," marks a paradigm shift in the design, deployment, and protection of data-intensive applications across the compute continuum. Building on robust security foundations established in earlier phases, this document introduces the overarching security framework that underpins the Data Broker component and its integration within the broader NEARDATA ecosystem.

Modern data processing applications, especially those that leverage artificial intelligence and real-time analytics, face unprecedented challenges in balancing performance, scalability, and security. The distributed nature of edge-to-cloud environments introduces a wide spectrum of attack vectors and privacy risks that cannot be adequately addressed by conventional centralised security models. NEARDATA addresses these challenges through an innovative combination of trusted execution environments (TEEs), federated processing, and privacy-preserving data flows, all designed to guarantee strong security assurances without undermining computational efficiency.

At the heart of NEARDATA's architecture lies the Data Broker, the central orchestration component responsible for enabling secure data exchange between heterogeneous data sources, processing engines, and consumer applications. Unlike traditional data management systems, the Data Broker enforces a zero-trust security paradigm, embedding data protection mechanisms throughout the entire processing pipeline. This approach is especially vital for sensitive domains such as healthcare analytics, where compliance with stringent regulatory frameworks and data sovereignty requirements requires uncompromising safeguards.

2.1 Target Audience

The present document, which evaluates the Data Broker security framework, is formally intended for internal circulation within the NEARDATA consortium, though it remains publicly accessible. Its primary audience comprises the NEARDATA technical team, including all partners engaged in work packages related to data management, security implementation, and platform integration. In addition, the document serves as a technical reference for developers working on NEARDATA use cases, with particular relevance for federated learning applications in the surgical and metabolomics domains. It also provides practical guidance for system architects and security engineers responsible for deploying NEARDATA components across cloud and edge environments.

2.2 Structure of document

This document includes four main parts:

This document is organized into the following key sections:

1. **Control Plane Architecture** This section details the architectural components of the Data Broker, including:
   (a) **Core Data Broker Design:** Foundational Architecture and its role in orchestrating secure data flows.
   (b) **Confidential Compute Layer:** Integration of TEEs and confidential computing to protect data in use, at rest, and in transit.
   (c) **Confidential Data Exchange Layer:** Mechanisms for secure, policy-driven data exchange between distributed sources and consumers.
   (d) **Orchestration Layer:** Management of data processing workflows, resource allocation, and secure service composition.
   (e) **Federated Orchestration Layer:** Coordination of federated learning and distributed analytics across multiple, potentially untrusted domains.

   This section explains how these layers interact to manage and secure data flows across the compute continuum, implementing a multi-layered security approach that aligns with best practices in modern data broker architectures

2. **NEARDATA Threat Model and Policy Model** This part provides a comprehensive summary of potential threats and attacks in the context of streaming data processing and AI applications across the compute continuum. It presents:
   (a) **Threat Model:** Analysis of attack vectors, including those specific to cloud, edge, and federated environments (e.g., data leakage, model inversion, poisoning, side-channel attacks).
   (b) **Policy Model:** Detailed policy definitions, including protection goals and mechanisms designed to address the analyzed threats within the Data Broker architecture. This includes access control, encryption, attestation, and auditing strategies, as well as compliance with regulatory requirements.
3. **Use Cases** This section demonstrates how the Data Broker security framework supports real-world applications, with detailed examples from:
   (a) **Surgical with Federated Learning and Pravega and Flower framework:** Showcasing secure streaming and federated learning for surgical data, leveraging Pravega for high-throughput video streaming and Flower for privacy-preserving distributed model training, all protected by SCONE's confidential computing.
   (b) **Metabolomics with Lithops, Metaspace, and Keycloak:** Detailing the application of the Data Broker in spatial metabolomics annotation, utilizing Lithops for multi-cloud serverless computing, Metaspace for annotation, and Keycloak for identity and access management. This use case highlights secure, scalable, and confidential processing of large-scale biomedical data.
4. **Final Release of the Data Broker** This section provides an overview of the final release of the Data Broker, highlighting its features and capabilities.
   (a) **Features and capabilities:** Secure data governance (Confidential data transfer, Policy-driven access control)
   (b) **Security Tools:** The suite of tools developed to control data access and provide confidentiality and integrity for applications running within NEARDATA, including the SCONE runtime (enabling applications to run in TEEs), a cross-compiler, and the Sconify tool (enabling developers to convert native apps into confidential ones)
5. **Performance Evaluation** This section presents the results of a preliminary evaluation, which was conducted to validate various design decisions and assess the performance of the system.
6. **Conclusion section**, We summarise our contributions to securing data brokering in extreme-scale processing environments, and we outline the ongoing and future work planned to enhance the platform's security capabilities.

The document concludes with a summary of achievements and outlines ongoing and future work planned for enhancing the platform's security capabilities.

3 Data Broker Reference Architecture

3.1 Control Plane: Confidential Data Orchestration

The NEARDATA Control Plane delivers robust confidential data orchestration across the entire Cloud-Edge continuum, ensuring secure governance, processing, and exchange of sensitive data in distributed environments. Leveraging trusted execution environments (TEEs) and advanced cryptographic mechanisms, the control plane enforces strict confidentiality and integrity guarantees, even in multi-tenant and federated learning scenarios where data must remain at its origin while enabling collaborative computation and model training across multiple stakeholders. This approach is fully aligned with the zero-trust security paradigm, which requires continuous verification of identity, context, and policy compliance for every access request, regardless of network location.

Core Functions of the Control Plane

1. **Orchestrating data flows and analytics pipelines across diverse environments**, ensuring that data is processed where it is most efficient and secure.

2. **Managing security, access control, and policy enforcement for data sharing and computation**, in line with modern data governance principles.

3. **Coordinating the deployment and execution of data processing tasks, including confidential and federated learning workflows**, while maintaining compliance with privacy regulations

3.1.1 Data Broker Architecture

Data Broker and Data Connectors are integral components of NEARDATA's architecture, and as such, their notions are innovatively redefined as pioneering improvements inside our Data Space. The Data Plane is a sophisticated data service that acts as an intermediary, enhancing and streamlining data flows by leveraging the S3 API for storing and retrieving objects, as well as streaming APIs via fast near-data connections in Cloud/Edge environments.

To further strengthen the trustworthiness and security of NEARDATA's Data Broker, we integrate SCONE (Secure Container Environment) as a foundational technology. SCONE is not merely a platform component within NEARDATA's architecture, it is a cornerstone for enabling trustworthy, confidential, and policy-driven data operations across the entire data lifecycle. Its significance extends far beyond providing a runtime environment:

Key Comparative Strengths

1. 1**End-to-End Confidentiality** SCONE ensures that data is encrypted not only at rest and in transit, but also during computation. This comprehensive approach means sensitive information remains protected throughout its entire lifecycle, even from privileged insiders like system administrators.

2. **Seamless Integration with Containers** A standout feature is SCONE's ability to secure existing containerized applications with little to no code modification. This allows organizations to adopt confidential computing without re-architecting their software, streamlining the path to enhanced security.

3. **Remote Attestation for Trust** SCONE provides robust remote attestation, enabling users to verify that their applications are running in a genuine, untampered secure enclave. This builds trust in the integrity of both the application and the underlying platform.

4. **Confidential Service Mesh Support** SCONE supports secure orchestration and communication between multiple microservices, making it suitable for complex, distributed applications that require confidential interactions across service boundaries.

5. **Broad Language and Application Support** Unlike solutions that focus only on compiled languages, SCONE can secure interpreted languages such as Python. This flexibility makes it ideal for a wide range of workloads, including data analytics and machine learning.

6. **Performance Optimization** SCONE employs advanced thread management and asynchronous system calls, which help maintain high performance and scalability, even for demanding, multi-threaded applications.

7. **Minimal Trusted Computing Base** By isolating only the application and its data within the secure enclave, SCONE reduces the amount of code that must be trusted, minimizing potential vulnerabilities and simplifying security audits.

Figure 1 illustrates the architecture of NEARDATA's Data Broker components and tooling we utilize in order to run applications in a trustworthy manner. The Data Broker's Confidential Compute Layer incorporates a CAS as a vital element within this framework. Furthermore, the Confidential Orchestration of the Data Broker is managed via Kubernetes. To enhance data security, we will also utilize SCONE's network shield within the Confidential Data Exchange component of the Data Broker, which is an integral part of the SCONE runtime, enabling precise control over data access.

On the left side of the figure, a list of tools is depicted which are used for transforming native applications into a confidential ones. The transformation can be either achieved through the SCONE cross compiler or the so-called sconify tool.

In the figure, components are depicted that come into play during the runtime of an application, assuming the hardware supports the execution of applications in TEEs.

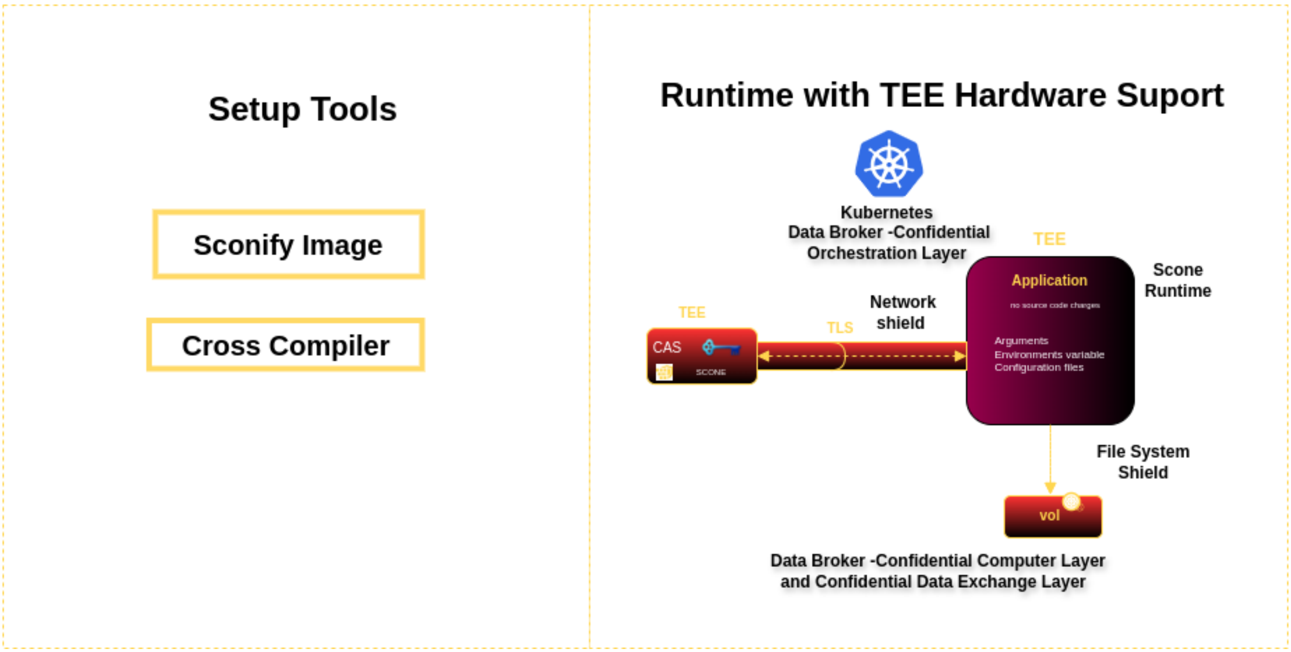


Figure 1: Security tools architecture.

The high-level policy definitions stored as Custom Resource Records (CRR) using Custom Resource Definitions (CRD) within the Kubernetes cluster which application user typically uses to deploy his or her application. This approach aligns with the cloud-native paradigm and represents the current standard for deploying modern applications. It's worth noting that the policy session language for the Data Brokers Configuration and Attestation Service (CAS) is stored in a separated CAS instance rather than in the etcd server of the Kubernetes cluster. In addition to storing records in the Kubernetes cluster, a Kubernetes operator will be developed to execute the necessary configuration steps required for deploying application within the context of NEARDATA securely and confidentially. The key tasks the Kubernetes operator will perform as follows:

- **Sconifies Docker Images:** Transforms native Docker images into confidential ones using the Sconify process.
- **Extracts and Converts Program Arguments and Environment Variables:** Gathers and converts program arguments and environment variables according to the defined security policies.
- **Generates SCONE Sessions:** Creates SCONE sessions based on the parsed information and submits them to the CAS for further processing.

This comprehensive set of steps ensures that the deployment process is not only aligned with the defined security policies but also takes advantage of confidential computing features provided by tools like SCONE within a Kubernetes environment. The Kubernetes operator acts as an automated orchestrator, seamlessly integrating security measures into the deployment workflow.

Key capabilities of the Data Broker include:

- **Zero Trust Data Governance Framework:** The Data Broker enforces comprehensive access control policies, supporting data residency, institutional sharing agreements, and regulatory compliance. Every data access or movement is subject to continuous authentication and authorization checks, in line with zero-trust principles . Integration with data catalogs ensures consistent governance throughout the data lifecycle, even across organizational boundaries.

- **Trustworthy Flow Orchestration:** Multi-party workflows are coordinated such that sensitive data never leaves authorized environments. All data movements are validated against governance policies, and cryptographic proofs of compliance are maintained, supporting auditability and regulatory requirements.

- **Heterogeneous Service Integration:** The broker abstracts the complexity of integrating diverse data processing services and TEE implementations, providing uniform security semantics across different hardware platforms. This enables seamless orchestration of confidential workloads on both edge and cloud resources.

By leveraging data access components and connectors from the data plane, the Data Broker creates a unified control interface that spans edge devices, institutional resources, and public clouds. This architecture enables organizations to maintain data sovereignty while participating in collaborative analytics and research, all under a zero-trust security model.

3.1.2 Confidential Compute Layer

The Confidential Compute Layer is foundational to NEARDATA's security, providing TEE-protected execution for sensitive data processing. It supports multiple TEE technologies, including Intel SGX, ARM CCA, and emerging GPU-based confidential computing, ensuring broad compatibility and flexibility.

- **Multi-Platform TEE Support:** The framework dynamically selects the optimal TEE implementation based on workload and hardware availability. For example, SCONE enables transparent enclave management, attestation, and secure communication for Intel SGX, while ARM CCA extends confidential computing to edge and mobile devices.

- **Transparent Application Transformation:** SCONE's binary transformation and runtime interposition allow existing applications to run confidentially without source code changes or recompilation. Sensitive operations are redirected through TEE-protected channels, preserving both compatibility and performance.

- **Verified Data Connector Execution:** Data connectors are executed within sealed environments, with cryptographic verification and attestation ensuring that data transformation and transfer cannot be observed or tampered with by unauthorized entities.

- **High-Volume Data Processing:** The layer is optimized for large-scale scientific and healthcare datasets, using advanced memory management and efficient cryptographic operations to minimize the performance overhead typically associated with confidential computing.

This layer is tightly integrated with the data plane, ensuring that all data processing benefits from TEE protections and that confidential computing is accessible across the NEARDATA ecosystem.

3.1.3 Confidential Data Exchange

The Confidential Data Exchange subsystem enables secure, end-to-end encrypted data transfer across network boundaries, supporting collaborative computation without exposing sensitive data during transit or at intermediate points.

- **Mutual Authentication Framework:** All participating systems undergo mutual authentication using cryptographic certificates and TEE attestation, ensuring that data is only exchanged with verified, authorized recipients operating in secure environments.

- **High-Performance Encrypted Transport:** Optimized cryptographic primitives, including hardware-accelerated encryption and pipeline processing, enable high-throughput, low-latency data transfers—critical for real-time analytics and federated learning.

- **Transparent Integration:** Applications benefit from confidential data exchange without code changes; the subsystem intercepts network operations and applies encryption, authentication, and authorization policies automatically.

- **Policy-Driven Access Control:** Fine-grained access policies govern all data exchanges, considering data classification, recipient identity, intended use, and temporal constraints. This supports complex, multi-stakeholder federated research scenarios.

- **Secure Storage Integration:** Data at rest is transparently encrypted, ensuring that even cloud providers or administrators cannot access stored data, in line with best practices for data sovereignty and privacy.

3.1.4 Confidential Orchestration

Confidential Orchestration extends platforms like Kubernetes with confidential computing capabilities, ensuring that entire application deployments operate within trusted environments while remaining compatible with standard DevOps and container workflows.

- **Zero Trust Kubernetes Integration:** The orchestration layer is TEE-aware and zero-trust compliant, scheduling confidential workloads on appropriate hardware and maintaining standard Kubernetes APIs. Every workload deployment and access request is continuously authenticated and authorized, regardless of network location or user role [1].

- **Identity Broker Services:** Federated identity management maintains trust relationships across institutional domains, leveraging both PKI and TEE attestation for strong authentication while preserving privacy and autonomy.

- **Tamper-Proof Application Deployment:** All applications are cryptographically verified before deployment, and runtime modifications trigger security alerts and potential termination, ensuring integrity throughout the application lifecycle.

- **Zero Trust Compliance:** The architecture is designed to enforce zero-trust principles at every layer, ensuring that no implicit trust is granted based on network location, device, or user identity. All access and data flows are subject to continuous verification and least-privilege enforcement.

- **Privacy-Aware Data Manipulation:** Data processing automatically adapts to governance policies, using techniques such as differential privacy, data masking, and selective disclosure to ensure compliance with regulatory and institutional requirements.

3.1.5 Federated Learning Orchestration

The Federated Learning Orchestration framework enables collaborative machine learning across institutional boundaries, allowing organizations to jointly train models without sharing raw data—a critical requirement for privacy-sensitive domains like healthcare.

- **Decentralized Training Architecture:** Training data remains at its origin, with local model training in TEE-protected environments. This ensures that raw data never leaves institutional boundaries, supporting both privacy and compliance.

- **Secure Model Aggregation:** Aggregation servers operate within attested TEEs, using advanced cryptographic protocols to combine local model updates while preserving the confidentiality of individual contributions and mitigating attacks such as model poisoning.
- **Zero Trust Federated Learning:** The orchestration framework applies zero-trust security to every stage of the federated learning process, continuously verifying the identity and authorization of all participants and enforcing strict access controls on model updates and aggregation.
- **Framework Compatibility:** Built on open-source federated learning platforms like Flower and FedML, the orchestration framework is compatible with popular ML frameworks (e.g., TensorFlow, PyTorch), facilitating adoption without significant code changes.
- **Flexible Infrastructure and Serverless Simulation:** The framework leverages NEARDATA's secure infrastructure and, as a key project contribution, extends the Flower platform with a novel, Lithops-based simulation engine. This integration introduces serverless computing as a highly scalable and flexible alternative to the existing Ray simulation engine for serverful infrastructure. It empowers researchers to conduct large-scale simulations on their infrastructure of choice—supporting federations with thousands of clients without the overhead of provisioning and managing dedicated clusters—while ensuring consistent security and governance across the entire lifecycle.[1]
- **Quality Assurance and Monitoring:** Confidential monitoring tracks training progress, detects anomalies, and provides audit trails for compliance, all within TEE-protected environments to ensure monitoring data remains confidential.

This federated learning orchestration is a key enabler for collaborative research, particularly in healthcare, where it allows institutions to jointly develop advanced models while maintaining strict privacy and regulatory compliance. The integration with SCONE and NEARDATA's confidential computing infrastructure ensures that all aspects of the federated learning process are protected by strong security guarantees and comprehensive governance, fully aligned with zero-trust security principles.

3.2 NEARDATA Security Overview

In the following section, we provide a comprehensive overview of the security-related components integrated into the NEARDATA platform. Table 1 summarizes the individual tools and subcomponents, outlining their specific security functions, technical capabilities, and the responsible development partners. This overview establishes the foundation for understanding how the platform systematically addresses confidentiality, integrity, and availability requirements across the entire compute continuum, from edge devices to cloud infrastructure.

The security architecture of NEARDATA is built upon a multi-layered approach that combines a hardware-based trusted execution environment and policy-driven access controls. Each component has been designed to operate both independently and as part of an integrated security ecosystem, ensuring comprehensive protection for sensitive data processing workflows while maintaining the performance characteristics required for extreme-scale analytics applications.

After presenting the individual security components, we provide a detailed analysis of the threat model specifically developed for NEARDATA's distributed processing environment. This threat model has guided the systematic derivation of security requirements and informed the definition of security policies tailored for different stakeholder categories, including data providers, application developers, and system operators. These high-level security policies represent fundamental requirements originating from platform users and regulatory frameworks, particularly those governing healthcare data processing and bioinformatics research.

To enable fine-grained security configuration and dynamic policy adaptation, the NEARDATA platform employs sophisticated policy management mechanisms and session descriptions that specify application-level security constraints, multi-party attestation workflows, and runtime protection parameters. The policy framework supports context-aware security decisions, allowing the system

[1]The contribution was submitted to the official Flower repository (`https://github.com/adap/flower/pull/5658`) and the full implementation is also available in the project's repository fork: `https://github.com/janprz/flower/tree/add-lithops-backend-support`.

to adapt protection mechanisms based on data sensitivity levels, processing locations, and trust relationships between federated participants.

Finally, we describe how the comprehensive security architecture integrates these policies across the Data Broker and other platform services, creating a unified security fabric that spans the entire data processing pipeline. This integration ensures consistent security enforcement while supporting the platform's core objectives of enabling secure federated learning, privacy-preserving analytics, and compliant data sharing across organizational boundaries.

Table 1: NEARDATA Security Components Overview - Core Infrastructure

Tool/Component	**Security Function & Technical Description**	**Lead / Major Contributors**
Data Broker Security Core	Central security orchestration service implementing zero-trust architecture for secure data stream exchange. Provides comprehensive policy enforcement, access control, and secure session management between distributed sources, processing engines, and consumer applications.	TUD, Consortium Partners
SCONE Runtime Environment v5.8+	Hardware-based trusted execution environment leveraging Intel SGX technology (SDK 2.18) for containerized workloads. Provides memory encryption with EPC exhaustion warnings, remote attestation with DCAP v4 support, and secure code execution guarantees.	TUD
CAS (Configuration and Attestation Service)	Centralized trust management service handling remote attestation procedures with MAA token support, secure application and configuration deployment.	TUD
Sconify Security Tool	Enhanced command-line interface tool for automated transformation of standard container images into SGX-enabled confidential containers.	TUD
Dynamic Policy Management Framework	Implements fine-grained, context-aware security policies for data flows and federated applications with governance access policy rules.	TUD, Consortium Partners
Keycloak Identity Integration	Enterprise-grade identity and access management system and role-based authorization integrated with Data Broker for secure user and application identity verification across federated deployments with TLS mutual authentication.	SCONTAIN
Secure Lithops Integration	Enhanced serverless computing framework with integrated policy enforcement and confidential execution support via SCONE's Kubernetes operator. Enables secure, scalable execution of data analytics functions within TEE-protected environments while maintaining serverless flexibility and performance characteristics.	SCONTAIN, TUD, Partners
Pravega-SCONE Integration	Secure, confidential data streaming with Pravega, leveraging SCONE for end-to-end encryption and integrity protection of data streams. Integrates a sconified Pravega client running inside a Trusted Execution Environment (TEE), ensuring confidentiality and integrity for both data in transit and at rest. Experimental results show affordable latency increase at low throughput (up to 2×), and negligible overhead at high throughput (e.g., 10,000 events/sec), making it practical for latency-sensitive, real-time analytics on sensitive data.	TUD, DELL,SCO

Table 2: NEARDATA Security Components Overview - Advanced Features & Applications

Tool/Component	Security Function & Technical Description	Lead / Major Contributors
Federated Learning Security Framework	Privacy-preserving collaborative machine learning infrastructure supporting secure multi-party computation with Flower ML integration. Implements differential privacy mechanisms, secure aggregation protocols, and TEE-based model training for sensitive applications including surgical use case .	Use Case Partners, TUD
Zero Trust Network Architecture	Comprehensive network security framework implementing continuous verification, micro-segmentation, and encrypted communications across the entire NEARDATA platform.	Network Security Partners, TUD

3.3 Threat Model

NEARDATA is an advanced near-data processing and edge computing platform designed for efficient ingestion, storage, and retrieval of data streams in distributed environments. Its modular and scalable architecture integrates components such as stream storage engines (e.g., Pravega), serverless computing frameworks (e.g., Lithops), and confidential computing technologies (e.g., SCONE). NEARDATA is typically deployed close to data sources—at the edge or near-edge—where resources are shared among multiple applications, tenants, and users. This architectural approach is consistent with recent surveys on near-data processing and computational storage, which highlight the performance and security benefits of processing data near its source, while also introducing new security challenges due to increased architectural complexity and resource sharing [2, 3].

NEARDATA environments are inherently multi-tenant, meaning that computational, storage, and network resources are shared among diverse users and applications. This shared model introduces risks: if a device or process is compromised, it could potentially access or disrupt resources belonging to other tenants. The distributed and heterogeneous nature of NEARDATA, combined with the evolving state of near data processing security best practices, means that consistent and comprehensive security measures are still maturing [4, 5].

Table 3: Summary of Key Threats to NEARDATA and Mitigation Strategies

Threat Category	Threat Description	Mitigation Strategy
Potentially Malicious Environment	Privileged processes (OS, container runtime, orchestration layer) may be compromised, allowing attackers to manipulate system calls, intercept data, and monitor activities. This expands the attack surface in edge and near-data processing environments.	Employ hardware-based Trusted Execution Environments (TEEs) such as Intel SGX to isolate sensitive computations from the host OS and hypervisor, ensuring even root-level attackers cannot access protected data or code.
Hardware Trust Boundary	Physical access to hardware is tightly controlled, but attackers may install privileged software (e.g., rootkits, malicious hypervisors) with administrative rights.	Data Broker performs remote attestation of enclaves before provisioning secrets or configuration, ensuring only trusted hardware and software stacks are used.

Continued on next page

Table 3 (Continued): Summary of Key Threats to NEARDATA and Mitigation Strategies

Threat Category	Threat Description	Mitigation Strategy
External Attackers	Adversaries may target data generated by field devices or edge nodes, especially if those devices lack robust access controls or encryption. Risk is heightened in distributed, multi-tenant edge environments.	Data Broker transparently encrypts data at rest, in transit, and in use. All network traffic between services is protected with TLS, with keys managed inside enclaves.
Data Confidentiality and Privacy	• **Privileged Data Exfiltration:** Attackers with superuser privileges can copy files or intercept data streams. Even with encryption, attackers may extract keys from memory dumps. • **Unauthorized Data Monetization:** Weak access controls or insufficient encryption can allow sensitive data streams to be monetized or sold in real time.	Data Broker enforces strict, policy-driven access to secrets and data. Only attested, authorized services can access protected resources. Cryptographic keys are managed within TEEs and never exposed in plaintext.

Continued on next page

Table 3 (Continued): Summary of Key Threats to NEARDATA and Mitigation Strategies

Threat Category	Threat Description	Mitigation Strategy
Data Integrity and Tampering	• **Malicious Data Injection:** Attackers may inject false or manipulated data into NEARDATA pipelines, corrupting analytics or AI model training. • **Training Code or Model Tampering:** Attackers with file system access can alter code or models used in analytics or AI workloads.	• Only attested, trusted data sources are allowed to participate in data pipelines. The Data Broker verifies the identity and integrity of each source before allowing data ingestion. Cryptographic verification and continuous monitoring are employed. • **CRISP (Confidentiality, Rollback, and Integrity Storage Protection)** is integrated as a last-level defense for application integrity and confidentiality, ensuring that data and code stored on untrusted media remain protected against tampering and rollback attacks. • **Last-Level Defense for Application Integrity and Confidentiality:** This approach provides an additional security layer at the storage boundary, ensuring that even if all other controls fail, the integrity and confidentiality of application data are preserved.
Availability and Service Disruption	• **Denial of Service (DoS):** Attackers may flood NEARDATA components with excessive requests, exhausting resources and disrupting legitimate services. • **Resource Exhaustion:** In multi-tenant settings, one tenant could monopolize resources, degrading performance for others.	Resource management, monitoring, and rate limiting are used to detect and mitigate DoS and resource exhaustion attacks. Isolation mechanisms prevent one tenant from impacting others.
Lateral Movement	Once a NEARDATA node is compromised, attackers may attempt to move laterally to access additional resources or data streams.	Data Broker supports namespaces and policy isolation, ensuring that compromise of one enclave or service does not grant access to others. Micro-segmentation and strict authentication are enforced.

Continued on next page

Table 3 (Continued): Summary of Key Threats to NEARDATA and Mitigation Strategies

Threat Category	Threat Description	Mitigation Strategy
Rollback and Replay Attacks	Attackers may supply outdated versions of data, models, or configurations, causing analytics or AI models to revert to less secure or less accurate states.	• Strong version control, cryptographic integrity checks, and attestation of data and model versions are used to prevent rollback and replay attacks. • CRISP provides a robust mechanism to detect and prevent rollback of stored data and application states, ensuring that only the latest, authorized versions are used.
Network and Communication Threats	Man-in-the-Middle (MitM) and injection attacks: Unsecured communication between NEARDATA components can be intercepted or manipulated, leading to data leakage or injection of malicious commands.	All communications are encrypted using TLS with mutual authentication. Keys are managed inside enclaves, and network protection shields (NPS) are used to ensure confidentiality and integrity in transit.
External Data Access and Tampering with Analytics/AI Workloads	• **External Data Access:** Insufficient or weak access controls on edge devices or APIs can be exploited to gain unauthorized access to, or exfiltrate, sensitive data streams. • **Tampering with Analytics/AI Workloads:** Attackers may inject false data, manipulate input streams, or alter intermediate results within analytics or AI workloads.	Robust authentication, authorization, and encryption mechanisms are implemented to protect data at the edge and during transmission. Data validation, integrity verification (e.g., cryptographic hashes or digital signatures), and continuous monitoring for anomalous behavior are employed throughout the analytics pipeline.

Continued on next page

Table 3 (Continued): Summary of Key Threats to NEARDATA and Mitigation Strategies

Threat Category	Threat Description	Mitigation Strategy
Micro-Architectural Side-Channel Attacks	Attackers exploit shared memory, caches, or row buffers in NDP modules to infer sensitive information, such as cryptographic keys or proprietary data streams.	• Isolation of sensitive workloads, monitoring for anomalous access patterns, and use of TEE features to minimize side-channel leakage. (Further research and adaptive defenses are recommended.) • **SinClave (Hardware-assisted Singletons for TEEs)** ensures that only a single, attested instance of a critical service runs within a TEE, reducing the risk of side-channel leakage from co-located malicious enclaves.
Black-Box Model Extraction and Inference Attacks	Attackers use probing and observation of system responses to infer internal states, extract AI/analytics models, or reconstruct sensitive data, even without detailed architectural knowledge.	Protect APIs with rate limiting, anomaly detection, and access controls. Monitor for systematic or suspicious query patterns.
Adversarial Attacks on AI Workloads	Attackers craft inputs to mislead models or extract proprietary algorithms, degrading the accuracy of edge AI models.	Employ lightweight, adaptive defenses such as input validation, adversarial training, and continuous monitoring for anomalous model behavior.
Supply Chain and Dependency Attacks	Attackers may compromise third-party components or dependencies to introduce vulnerabilities or backdoors.	• SCONE supports Software Bill of Materials (SBOM) and cryptographic integrity verification for all dependencies. Only verified and attested components are deployed. • **TICAL (Trusted and Integrity-protected Compilation of AppLications)** ensures that the entire compilation process, from source code to binary, is integrity-protected and verifiable, mitigating risks from compromised compilers or build environments.

3.4 Policy Definitions

In NEARDATA, application developers and users can describe their security constraints through a comprehensive policy framework that integrates with SCONE's Configuration and Attestation Service (CAS). The NEARDATA Data Broker employs SCONE's session language to define protection goals and mechanisms, enabling fine-grained control over data confidentiality, integrity, and access controls across distributed computing environments.

Users can specify their security requirements through two complementary approaches: selecting high-level protection goals that automatically map to appropriate security mechanisms, or directly specifying concrete mechanisms when detailed control is required. This dual approach accommodates both security-aware users who prefer abstracted goals and experts who need granular control over the underlying security infrastructure.

In NEARDATA, application developers and users define security constraints through a comprehensive policy framework integrated with SCONE's Configuration and Attestation Service (CAS). This framework not only enables fine-grained control over data confidentiality, integrity, and access controls across distributed environments, but also facilitates compliance with real-world standards such as **GDPR** (General Data Protection Regulation) and **HIPAA** (Health Insurance Portability and Accountability Act).

Compliance Mapping:

- **GDPR:** NEARDATA policies enforce data minimization, purpose limitation, and strong access controls, ensuring that only authorized entities can access or process personal data. Encryption at rest and in transit, as well as audit logging, support GDPR Articles 5, 25, and 32 [2].
- **HIPAA:** The framework ensures confidentiality, integrity, and availability of protected health information (PHI) through technical safeguards such as encryption, access control matrices, and audit trails, aligning with HIPAA Security Rule requirements (§164.312) [3]

3.4.1 Protection Goals and Mechanisms

The NEARDATA security framework defines six primary protection goals, each mapped to specific SCONE-based mechanisms as shown in Table 4. These mappings leverage SCONE's comprehensive confidential computing capabilities, including Trusted Execution Environments (TEEs), file system protection shields (FSPF), and network protection shields (NPS).

For environments lacking TEE support (such as GPU accelerators), SCONE provides fallback mechanisms including secure boot verification and platform-based attestation. Platform-based attestation allows SCONE CAS to establish trust in the execution environment by verifying the integrity of the boot process, firmware versions, and platform configuration, providing a foundation for confidential computing even without hardware TEE support.

[2] `https://gdpr-info.eu/art-5-gdpr/`

[3] `https://www.govinfo.gov/content/pkg/CFR-2004-title45-vol1/pdf/CFR-2004-title45-vol1-sec164-312.pdf.`

Attack	Mechanism
Memory dumps, i.e., stealing secrets and data temporarily stored in RAM (**confidentiality protection** for data being processed). Modifications of data structures stored in RAM (**integrity protection** for data being processed). Integrity protection for application.	Data Broker – Confidential Compute Layer (Trusted Execution Environment, TEE)
Reading & modifying input data for training as well as training models (**confidentiality protection** and **integrity protection** for data at rest).	Data Broker – Confidential Data Exchange Layer (File System Protection Shield, FSPS)
Reading data exchanged over the network during training (**confidentiality protection** and **integrity protection** for data in transit).	Data Broker – Confidential Data Exchange Layer (Network Protection Shield, NPS)
Reading & modifying or forging input data for training as well as training models (**confidentiality protection**, **integrity protection**, **lateral movement**, and **tampering with training results** for data at rest).	Mobile Device Authentication

Table 4: Attacks and corresponding security mechanisms

3.4.2 SCONE Session Language Integration

NEARDATA leverages SCONE's session language version 0.3.11 to define comprehensive security policies. The session language provides a YAML-based syntax for specifying services, secrets, attestation policies, and access controls. The following example demonstrates a NEARDATA federated learning configuration with mutual authentication:

```yaml

name: NEARDATA_FEDERATED_LEARNING
version: "0.3.11"

security:
  attestation:
    mode: hw
    tolerate: [outdated-tcb]
    ignore_advisories: []
  singleton_enclaves: true

services:
  - name: flower_server
    mrenclave: "a1b2c3d4e5f67890abcdef1234567890abcdef1234567890abcdef1234567890"
    command: python3
    args: ["/app/server.py"]
    environment:
      SCONE_HEAP: "1G"
      FLOWER_SERVER_ADDRESS: "0.0.0.0:8080"
    fspf_path: /app/fspf
    fspf_key: "$$SCONE::fspf_key$$"

  - name: flower_client
    mrenclave: "b2c3d4e5f67890abcdef1234567890abcdef1234567890abcdef123456789012"
    command: python3
    args: ["/app/client.py"]
    environment:
      SCONE_HEAP: "2G"
```

```yaml

volumes:
  - name: shared_models
    path: /app/models
    fspf_path: /models.fspf
    fspf_key: "$$SCONE::model_key$$"

images:
  - name: flower_base_image
    volumes:
      - name: shared_models
        path: /app/models

secrets:
  - name: fspf_key
    kind: aes-gcm-key
  - name: model_key
    kind: aes-gcm-key

  # Certificate Authority for mutual authentication
  - name: ca_private_key
    kind: ed25519-key
  - name: ca_certificate
    kind: x509-ca
    private_key: ca_private_key
    common_name: "NEARDATA_FL_CA"

  # Server certificates
  - name: server_private_key
    kind: ed25519-key
  - name: server_certificate
    kind: x509
    private_key: server_private_key
    issuer: ca_certificate
    common_name: "flower_server"
    endpoint: server
    dns: ["localhost", "flower-server.neardata.svc.cluster.local"]

  # Client certificates
  - name: client_private_key
    kind: ed25519-key
  - name: client_certificate
    kind: x509
    private_key: client_private_key
    issuer: ca_certificate
    common_name: "flower_client"
    endpoint: client

```

This configuration demonstrates several key SCONE features integrated into NEARDATA:

- **Hardware-based Attestation**: The `mode: hw` setting ensures applications run only in genuine Intel SGX enclaves.
- **Singleton Enclaves**: Prevents replay attacks and ensures fresh enclave instances.
- **MRENCLAVE Verification**: Each service specifies its expected enclave measurement for integrity verification.
- **File System Protection**: FSPF with AES-GCM encryption protects model data and application files.

- **Automatic Certificate Generation**: SCONE CAS generates and provisions TLS certificates for mutual authentication.
- **Secret Management**: Cryptographic keys are generated within TEEs and never exposed in plaintext.

3.4.3 Fine-Grained Policy Definitions

Beyond high-level protection goals, NEARDATA supports fine-grained policy definitions that enable complex multi-stakeholder scenarios. This is particularly important for federated learning where different participants may have varying trust relationships and data sharing requirements.

The SCONE session language supports advanced features including:

Secret Import and Export: Policies can import secrets from other sessions, enabling secure collaboration between different NEARDATA deployments while maintaining cryptographic separation of concerns.

Platform-based Attestation: For environments without TEE support, SCONE can verify platform integrity through measurements of the boot process, allowing confidential computing on a broader range of hardware.

Policy Fragments: Reusable policy components that can be imported across sessions, enabling standardization of security configurations across NEARDATA deployments.

Access Control Matrices: Fine-grained permissions that specify which entities can read, update, or create sessions within specific namespaces.

Example fine-grained access control specification:

```

access_policy:
  read:
    - "$$SCONE::hospital_admin_cert$$"
    - "$$SCONE::researcher_cert$$"
  update:
    require-at-least-2:
      - "$$SCONE::hospital_a_admin$$"
      - "$$SCONE::hospital_b_admin$$"
      - "$$SCONE::hospital_c_admin$$"
  create_sessions:
    require-all:
      - "$$SCONE::data_protection_officer$$"
      - require-at-least-1:
          - "$$SCONE::lead_researcher$$"
          - "$$SCONE::ethics_board_chair$$"

```

This access control policy ensures that:

- Medical data can be read by authorized hospital administrators and researchers
- Policy updates require approval from at least two hospital administrators
- New federated learning sessions require both data protection officer approval and either lead researcher or ethics board authorization

3.5 Governance and Policy Board

NEARDATA leverages SCONE's advanced governance mechanisms to support multi-stakeholder decision making in federated learning environments. The governance framework addresses the challenge of distributed trust in healthcare scenarios where multiple hospitals must collaborate while maintaining institutional autonomy.

3.5.1 Multi-Signature Policy Management

SCONE's governance model enables policy boards where critical decisions require approval from multiple stakeholders. This is implemented through cryptographic signatures rather than traditional voting mechanisms, ensuring that governance decisions are tamper-proof and auditable.

The governance framework supports several approval patterns:

Threshold Signatures: Require at least N out of M stakeholders to approve changes. **Hierarchical Approval**: Different stakeholders have different approval authorities. **Veto Powers**: Specific stakeholders can block changes regardless of other approvals. **Automated Policy Checkers**: SCONE-protected programs that can automatically approve simple changes.

Example governance configuration for hospital federation:

```

access_policy:
  read:
    - "$$SCONE::ethics_board_cert$$"
    - "$$SCONE::data_protection_officer$$"
  update:
    require-at-least-1:
      - signer: "$$SCONE::medical_director_signer$$"
      - require-all:
          - require-at-least-3:
              - signer: "$$SCONE::hospital_a_signer$$"
              - signer: "$$SCONE::hospital_b_signer$$"
              - signer: "$$SCONE::hospital_c_signer$$"
              - signer: "$$SCONE::hospital_d_signer$$"
              - signer: "$$SCONE::hospital_e_signer$$"
          - require-all:
              - signer: "$$SCONE::privacy_officer_signer$$"
              - signer: "$$SCONE::security_officer_signer$$"

```

This policy ensures that federated learning model updates require either:

- Approval from the medical director (emergency override), or
- Approval from at least 3 out of 5 participating hospitals AND both privacy and security officers

3.5.2 Automated Policy Validation

NEARDATA includes automated policy checkers implemented as SCONE-protected services that can validate policy changes against regulatory requirements and best practices. These checkers run within TEEs to prevent tampering and can automatically approve routine changes while flagging complex modifications for human review.

Automated policy checkers validate:

- Compliance with GDPR, HIPAA, and other healthcare regulations.
- Adherence to institutional data sharing agreements.
- Consistency with established security baselines.
- Proper attestation policy configurations.
- Resource allocation limits and quotas.

The integration of governance, technical controls, and regulatory compliance ensures that NEARDATA provides a comprehensive platform for confidential federated learning in healthcare environments while maintaining the flexibility needed for research and clinical applications.

3.6 Advanced Access Control Mechanisms

NEARDATA extends SCONE's access control capabilities to support complex multi-tenant scenarios common in federated learning and distributed data processing environments. The access control framework operates at multiple levels: session-level permissions, service-level attestation, and data-level encryption.

3.6.1 Certificate-Based Authentication

The NEARDATA access control system relies on X.509 certificate-based authentication integrated with SCONE's attestation framework. Each stakeholder in a federated learning deployment possesses unique certificates that are verified both cryptographically and through enclave attestation.

Certificate validation occurs through multiple mechanisms:

Static Certificate Verification: Public keys are embedded in session policies and verified during TLS handshake **Dynamic Certificate Chains**: Certificate authorities can be managed within SCONE sessions, enabling certificate rotation **Attestation-Bound Certificates**: Certificates are only valid when presented by attested enclaves with specific measurements **Role-Based Certificate Extensions**: X.509 extensions carry role information validated by SCONE CAS

Example certificate hierarchy for hospital federation:

```

secrets:
  # Root CA managed by medical ethics board
  - name: ethics_board_ca_key
    kind: rsa-key
    size: 4096
  - name: ethics_board_ca_cert
    kind: x509-ca
    private_key: ethics_board_ca_key
    common_name: "NEARDATA Medical Ethics Authority"
    validity: "3650d" # 10 years

  # Intermediate CA for each hospital
  - name: hospital_a_intermediate_key
    kind: rsa-key
    size: 2048
  - name: hospital_a_intermediate_cert
    kind: x509-ca
    private_key: hospital_a_intermediate_key
    issuer: ethics_board_ca_cert
    common_name: "Hospital A Federated Learning Authority"
    validity: "1825d" # 5 years

  # End-entity certificates for specific roles
  - name: hospital_a_researcher_key
    kind: rsa-key
    size: 2048
  - name: hospital_a_researcher_cert
    kind: x509
    private_key: hospital_a_researcher_key
    issuer: hospital_a_intermediate_cert
    common_name: "Dr. Alice Johnson - Hospital A Researcher"
    validity: "365d" # 1 year
    extensions:
      - key: "1.3.6.1.4.1.12345.1" # Custom OID for role
        value: "researcher:surgical_oncology"

```

3.6.2 Multi-Level Security Integration

For highly sensitive medical data, NEARDATA supports integration with Multi-Level Security (MLS) frameworks. This enables fine-grained data classification where individual data elements can have different security levels, and access decisions are made based on both user clearance and data classification.

The MLS integration leverages SCONE's secret management capabilities to implement:

- **Security Labels**: Each data element carries cryptographically protected security labels
- **Clearance Verification**: User clearances are verified through attestation and certificate validation
- **Information Flow Control**: SCONE policies prevent information from flowing from high to low security levels
- **Trusted Computing Base**: All security decisions are made within attested enclaves

3.7 Integration with External Security Infrastructure

NEARDATA's policy framework is designed to integrate with existing enterprise security infrastructure while maintaining the security guarantees provided by confidential computing. This integration is crucial for healthcare organizations that must maintain compliance with existing security policies and regulatory frameworks.

3.7.1 Continuous Compliance Monitoring

NEARDATA implements automated compliance monitoring through SCONE-protected policy checkers that continuously validate system configuration against regulatory requirements. These monitors can detect policy drift, unauthorized access attempts, and potential compliance violations in real-time.

Compliance monitoring capabilities include:

- **Policy Validation**: Automated verification that all policies meet regulatory requirements
- **Access Pattern Analysis**: Detection of unusual data access patterns that might indicate policy violations
- **Encryption Verification**: Continuous monitoring that all sensitive data remains encrypted at all times
- **Attestation Monitoring**: Verification that all confidential workloads maintain proper attestation status

3.8 Performance and Scalability Considerations

The NEARDATA policy framework balances security requirements with the performance needs of large-scale federated learning and data processing workloads. Performance optimization focuses on minimizing the overhead of cryptographic operations and attestation while maintaining security guarantees.

3.8.1 Policy Evaluation Optimization

Complex access control policies can introduce significant overhead in high-throughput data processing scenarios. NEARDATA addresses this through several optimization strategies:

Policy Caching: Frequently evaluated policy decisions are cached within enclaves to avoid repeated cryptographic operations.

Batch Attestation: Multiple service instances can be attested collectively to reduce individual attestation overhead.

Lazy Evaluation: Complex policy conditions are evaluated only when necessary, with early termination for obvious allow/deny decisions.

Hardware Optimization: SGX-optimized cryptographic libraries minimize the performance impact of encryption and attestation operations.

3.8.2 Scalability Architecture

Large-scale federated learning deployments require policy frameworks that can scale to hundreds of participating institutions and thousands of concurrent workloads. NEARDATA addresses scalability through:

Hierarchical Policy Management: Policies are organized hierarchically to enable efficient delegation and reduce central bottlenecks.

Distributed CAS Deployment: Multiple SCONE CAS instances can be federated to distribute policy evaluation load.

Edge Policy Enforcement: Simple policy decisions can be made at edge nodes without central coordination.

Asynchronous Audit: Audit logging is performed asynchronously to avoid impacting real-time policy decisions.

3.9 Future Extensions and Research Directions

The NEARDATA policy framework establishes a foundation for advanced confidential computing applications in healthcare and beyond. Several areas of ongoing research and development extend the current capabilities:

3.9.1 Zero-Knowledge Policy Proofs

Advanced privacy requirements in multi-stakeholder environments may require zero-knowledge proofs of policy compliance without revealing the actual policies. NEARDATA is exploring:

- Zero-knowledge proofs of access control compliance
- Privacy-preserving audit mechanisms
- Selective disclosure of policy attributes

- Verifiable computation of policy decisions

The comprehensive policy framework described in this section provides the foundation for secure, compliant, and scalable federated learning and distributed data processing in healthcare environments. The integration of SCONE's confidential computing capabilities with NEARDATA's domain-specific requirements creates a robust platform for advancing medical research while maintaining the highest standards of patient privacy and data security.

4 Use Cases Enhanced through Data Broker Integration

4.1 Use case: surgery

The challenge of advancing surgical AI while maintaining patient privacy has found an innovative solution through federated learning a distributed machine learning approach that fundamentally changes how medical institutions collaborate on AI development. Instead of traditional methods that require centralized data collection, federated learning enables hospitals to train shared AI models by keeping sensitive surgical data within their own secure environments. During laparoscopic procedures, Pravega's streaming platform captures and preprocesses video data locally at each hospital, creating institutional datasets that never leave their originating facilities.

The Flower ML framework coordinates the federated learning process across participating hospitals, allowing them to collectively build sophisticated surgical AI models through a secure exchange of model parameters and training updates rather than raw patient data. This federated learning methodology ensures that each hospital's laparoscopic video recordings remain completely private while still contributing to a collaboratively trained model that benefits from the diverse surgical experiences across multiple institutions. SCONE's trusted execution environment provides comprehensive security throughout the entire federated learning pipeline, protecting both the local video processing at each hospital and the secure model parameter exchanges between institutions. Through this federated learning architecture, hospitals can harness the collective intelligence of their combined surgical expertise to develop more robust and generalizable AI models for applications such as surgical skill assessment, procedure optimization, and real-time surgical guidance, all while maintaining strict compliance with healthcare privacy regulations.

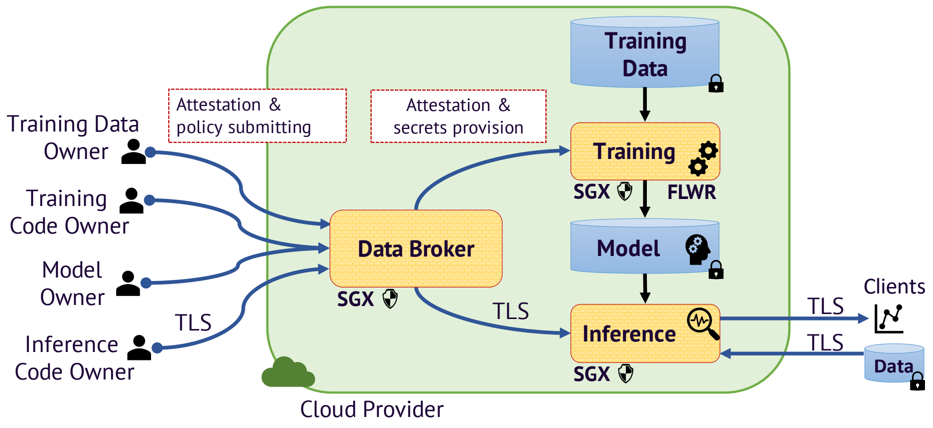


Figure 2: Federated Learning Architecture in NEARDATA.

Figure 2 illustrates the architecture of the federated learning application in NEARDATA. The main goal of our hardened version is not only to ensure the confidentiality, integrity and freshness of input data, code, and machine learning models but also to enable multiple clients (who do not necessarily trust each other) to get the benefits of collaborative training without revealing their local training data. In the confidential setup, each client performs the local training also inside TEE enclaves to make sure that no one tampers with the input data or training code during the computations. To govern and coordinate the collaborative machine learning training computation among clients, we design a Data Broker component which consists of a CAS which maintains security policies based on the agreement among all clients to define the access control over global training computation, the global training model, also the code and input data used for local training at each client. The CAS component transparently performs remote attestation to make sure the local computations are running the correct code, correct input data, and on the correct platforms as per the agreement. It only allows clients to participate in the global training after successfully performing the remote attestation process. It also conducts the remote attestation on the enclaves that execute the global training in a cloud, to ensure that no one at the cloud provider side modifies the global training aggregation computation. In addition to remote attestation, it encrypts the training code, and the CAS only provides the key to decrypt it inside enclaves after the remote attestation. Secrets including keys for encryption/decryption in each policy are generated by the Configuration and Attestation Service also running inside Intel SGX enclaves and cannot be seen by any human or client.

After receiving the agreed security policies from clients, the Data Broker CAS component strictly enforces them. It only passes secrets and configuration to applications (i.e., training computations), after attesting them. The training computations are executed inside Intel SGX enclaves and associated with policies provided and pre-agreed by clients. The training computations are identified by a secure hash and the content of the files (input data) they can access. Secrets can be passed to applications as command-line arguments, or environment variables, or can be injected into files. The files can contain environment variables referring to the names of secrets defined in the security policy. The variables are transparently replaced by the value of the secret when an application permitted to access the secrets reads the file. We design the CAS in a way that we can delegate its management of it to an untrusted party, e.g., a cloud provider, while clients can still trust that their security policies for protecting their properties are safely maintained and well protected. In the confidential version of the federated learning application, clients can attest to the Configuration and Attestation Service component, i.e., they can verify that it runs the expected unmodified code, in a correct platform before uploading security policies.

We have implemented the federated learning prototype jointly with NCT using FLOWER a distributed federated machine learning framework. Flower is a novel end-to-end federated learning framework that enables a more seamless transition from experimental research in simulation to system research on a large cohort. In order to showcase the features of our novel Data Broker, we performed the following steps:

1. The federated learning demo (using Flower) is executed/running inside of Intel SGX enclaves using SCONE, a framework to enable unmodified applications to run inside SGX enclaves.
2. We complemented the demo using SCONE's network shield, configured through the Data Broker's CAS component which protects network connections between client and server as well as controls data access.
3. In order to protect data at rest, we furthermore utilized SCONE's file system shield mechanism which encrypts the input (training) data using the file system shield of SCONE, and de-crypts it when processing inside of SGX enclaves.

As for the implementation of the Data Broker's CAS component, we partially rely on previous works.

This deliverable explains the selected use cases for federated learning, which serves to show the accomplishments of NEARDATA's objectives. The Data Broker component addressed objective O-3, which has been verified through the use of Key Performance Indicators (KPIs) 4. The objective is to ensure the secure coordination, transmission, manipulation, and retrieval of data. The third goal is to establish a Data Broker service that facilitates reliable data sharing and secure coordination of data pipelines throughout the Compute Continuum. The project aims to enhance the security and reliability of the data by using TEEs within federated learning systems. Policy-driven mutual authentication techniques also enable advanced data access policies for parties that do not trust each other. These policies are managed by policy boards, allowing access rules to be dynamically changed by human decision-makers and other entities.

The system employs encryption to secure both the input training data and code, such as Python code. It carries out all training calculations within TEE enclaves, including local and global training. This secure federated learning enables transmitting all gradient changes using TLS connections between the enclave of clients and the enclaves of the central training computation. Consequently, individuals who have elevated access cannot compromise the integrity and confidentiality of the input training data, code, and models. The Data Broker component is connected with CAS and utilises the remote attestation technique enabled by TEEs. This component guarantees the accuracy and reliability of the input data and training code. Put simply, it guarantees that training calculations are executed using accurate code, and accurate input data, and have not been tampered with by any unauthorised individuals, such as an attacker or malicious client. Based on the initial assessment, we can guarantee the privacy and security of federated learning calculations without compromising the effectiveness or accuracy of the training calculations.

The source codes for these use cases are publicly available.

Components	**Description**	**GitHub URL**
Surgery use-case	Federated Learning Source Code	`https://github.com/neardata-eu/nct_tud_fl_demo`

Table 5: Use-cases source codes released.

4.2 Use case: metabolomics

Spatial metabolomics annotation presents one of the most computationally demanding challenges in modern bioinformatics, requiring the processing of gigabytes of mass spectrometry imaging data to identify and map metabolite distributions within biological tissues. This field is critical for advancing drug discovery, disease research, and personalized medicine, as it provides unprecedented insights into how metabolites are spatially organized in complex biological systems. The computational complexity stems from the need to analyze thousands of molecular features across spatial coordinates, requiring sophisticated algorithms like those implemented in Metaspace to accurately detect metabolite structures and distinguish them from background noise and artifacts.

Lithops revolutionizes spatial metabolomics processing by implementing a multi-cloud serverless computing approach that breaks down massive annotation tasks into manageable, parallelizable computation slices distributed across cloud clusters. This framework enables researchers to leverage the elastic scalability of cloud computing, dynamically allocating resources based on workload demands while significantly reducing processing times for complex metabolomics analyses. MinIO provides the essential storage infrastructure through its S3-compatible object storage system, serving both as a secure repository for raw mass spectrometry files and as a coordination hub for pipeline execution, ensuring seamless data synchronization across distributed compute nodes.

SCONE's integration across the entire processing stack creates a comprehensive confidential computing environment that protects sensitive biological data and proprietary research algorithms throughout the cloud pipeline. This security framework supports Lithops execution across multiple cloud platforms through confidential containers for Kubernetes and Singularity compute backends, while simultaneously securing MinIO's storage operations. This end-to-end protection is particularly crucial for pharmaceutical research and clinical applications where metabolomics data may contain patient information or represent valuable intellectual property, enabling researchers to harness powerful cloud computing resources while maintaining strict data confidentiality and regulatory compliance.

4.2.1 Lithops

Lithops has a new backend named "*Singularity*", that introduces a new engine for working with "*threads*" instead of the traditional OS processes "*forks*". It requires `Python 3.13t`, that implements improvements on parallelization of threads[4] with the optional disabling of GIL[5].

The setup requires a *RabbitMQ* server to concentrate the synchronization between submitters (client application) and workers (Lithops Singularity instances). Clients will submit work to a queue in RabbitMQ and Lithops Singularity instances will read that queue and process accordingly. Example of a small Lithops program execution using Singularity backend deployed on a Kubernetes cluster:

-
  ```
  $ SCONE_VERSION=1 SCONE_CONFIG_ID=horizon-mesh-20135-21162/lithops/multiproc \
  SCONE_MODE=HW SCONE_LOG=ERROR SCONE_HEAP=3G PYTHON_GIL=0 \
  python3.13 multiprocessinglithops.py
  [SCONE] Using the following values (default values or specified in the untrusted
  environment):
  SCONE_QUEUES=4
  SCONE_SLOTS=256
  SCONE_SIGPIPE=0
  SCONE_MMAP32BIT=0
  ...
  Enclave hash: 3170dfd56a81dfc0508c37be6e44519bcd4123c6a928978abd7e0a6ed5212f8a
  2025-09-09 12:38:04,427 [INFO] config.py:139 - Lithops v3.6.1.dev0 - Python3.13
  2025-09-09 12:38:05,494 [INFO] minio.py:62 - MinIO client created - Endpoint:
  http://10.36.152.162:9000
  2025-09-09 12:38:06,080 [INFO] singularity.py:55 - Singularity client created
  2025-09-09 12:38:06,081 [INFO] invokers.py:119 - ExecutorID 83d2d8-0 | JobID M000 - Selected
  Runtime: singularity-metabolomics1.sif - 512MB
  2025-09-09 12:38:06,260 [INFO] invokers.py:186 - ExecutorID 83d2d8-0 | JobID M000 - Starting
  function invocation: cloud_process_wrapper() - Total: 4 activations
  2025-09-09 12:38:06,261 [INFO] invokers.py:225 - ExecutorID 83d2d8-0 | JobID M000 - View
  execution logs at /tmp/lithops-root/logs/83d2d8-0-M000.log
  2025-09-09 12:38:06,263 [INFO] executors.py:494 - ExecutorID 83d2d8-0 - Getting
  ```

[4]What's New In Python 3.13 https://docs.python.org/3/whatsnew/3.13.html
[5]GIL – global interpreter lock https://docs.python.org/3/glossary.html#term-global-interpreter-lock

```
results from 4 function activations
2025-09-09 12:38:06,263 [INFO] wait.py:101 - ExecutorID 83d2d8-0 - Waiting for 4
function activations to complete

100%|......................................................................| 4/4

2025-09-09 12:38:10,500 [INFO] executors.py:618 - ExecutorID 83d2d8-0 - Cleaning
temporary data
[2, 4, 6, 8]
```

Lithops's traditional working mechanics (considering `backend: localhost` and `backend: k8s`) is to divide the work using OS processes and Containers plus OS processes on a cluster, respectively. In confidential computing, this is very expensive, because for each OS process `fork()`'ed and container created a full copy of enclave's memory is made to the new instance. For example, if Lithops requires `runtime_memory: 768` MB, SCONE has to reserve `enclave=1024` MB; if `runtime_memory: 1280` MB, then `enclave=2048` MB is reserved; if `runtime_memory: 3072` MB, then `enclave=4096` MB is reserved; and so on exponentially in powers of 2. Lithops Singularity saves resources by not having to spawn dozens of containers on a cluster; instead, one or multiple instances can be started to process the user's data using less resources.

4.2.2 Metaspace

Metaspace is a Python system specialized in processing metabolite annotation of imaging mass spectrometry data and spatial metabolites. It employs Lithops as an engine to distribute workload among backend nodes in the cloud[6]. Metaspace, *as-is*, is exclusively compatible with Python 3.8; and this poses a challenge to the new approach of Lithops Singularity, that requires Python 3.13t. To cope with that, URV forked the Metaspace repository from Github and tailored an adapted version for Python 3.13[7]. The setup consists of 6 instances of Lithops Singularity, 1 instance of MinIO, and 1 instance of RabbitMQ serving 1 client using their services. Here is an excerpt from the sconified execution of the demonstration metabolomics pipeline:

-
  ```
  $ SCONE_VERSION=1 SCONE_CONFIG_ID=horizon-mesh-20135-21162/lithops/mboltest \
  SCONE_HEAP=7G python3.13 test-local.py
  [SCONE] Using the following values (default values or specified in the untrusted
  environment):
  SCONE_QUEUES=4
  SCONE_SLOTS=256
  SCONE_SIGPIPE=0
  ...
  SCONE_MPROTECT=no
  SCONE_FORK=yes
  SCONE_FORK_OS=1
  SCONE_CONFIG_ID: horizon-mesh-20135-21162/lithops/mboltest
  musl version:  1.2.5
  SCONE version:  5.9.0-239-g941a371d1-dirty-sergei/lithops (base:  f9230ace) (2025-01-20
  14:09:47)
  Enclave hash:  d5d66f2c6fac1e4f68d3199636889feb87f324ef1a899ec97ea55699eba3717d
  2025-05-05 14:16:25,548 [INFO] config.py:139 - Lithops v3.6.1.dev0 - Python3.13
  2025-05-05 14:16:25,548 [DEBUG] config.py:179 - Loading Serverless backend module:
  singularity
  2025-05-05 14:16:25,555 [DEBUG] config.py:219 - Loading Storage backend module:
  minio
  2025-05-05 14:16:25,562 [DEBUG] minio.py:38 - Creating MinIO client
  2025-05-05 14:16:25,562 [DEBUG] minio.py:43 - Setting MinIO endpoint to
  http://10.36.152.162:9000
  2025-05-05 14:16:26,280 [INFO] minio.py:62 - MinIO client created - Endpoint:
  http://10.36.152.162:9000
  2025-05-05 14:16:26,288 [DEBUG] singularity.py:41 - Creating Singularity client
  2025-05-05 14:16:26,381 [INFO] singularity.py:55 - Singularity client created
  ...
  ```

[6]Serverless METASPACE with Lithops https://github.com/metaspace2020/Lithops-METASPACE
[7]METASPACE annotation pipeline adaptation to python 3.13 – https://github.com/GEizaguirre/metaspace-py13/

```
2025-05-05 14:21:33,625 [DEBUG] monitor.py:144 - ExecutorID 7e88c3-0 - Pending:
0 - Running:  0 - Done:  32
2025-05-05 14:21:33,657 [DEBUG] monitor.py:465 - ExecutorID 7e88c3-0 - Storage
job monitor finished
2025-05-05 14:21:34,611 [DEBUG] future.py:229 - ExecutorID 7e88c3-0 | JobID M012
- Got status from call 00022 - Activation ID: 5e202a2b440f - Time:  9.49 seconds
2025-05-05 14:21:34,614 [DEBUG] future.py:286 - ExecutorID 7e88c3-0 | JobID M012
- Got output from call 00022 - Activation ID: 5e202a2b440f
...
2025-05-05 14:22:00,345 [DEBUG] monitor.py:144 - ExecutorID 7e88c3-0 - Pending:
0 - Running:  0 - Done:  1
2025-05-05 14:22:00,367 [DEBUG] monitor.py:465 - ExecutorID 7e88c3-0 - Storage
job monitor finished
2025-05-05 14:22:01,295 [DEBUG] future.py:229 - ExecutorID 7e88c3-0 | JobID M013
- Got status from call 00000 - Activation ID: bc6931c690b9 - Time:  1.49 seconds
2025-05-05 14:22:01,298 [DEBUG] future.py:286 - ExecutorID 7e88c3-0 | JobID M013
- Got output from call 00000 - Activation ID: bc6931c690b9
2025-05-05 14:22:01,311 [INFO] executors.py:618 - ExecutorID 7e88c3-0 - Cleaning
temporary data
2025-05-05 14:22:14,539 [DEBUG] executors.py:519 - ExecutorID 7e88c3-0 - Finished
getting results
```

Proceeding to the next step of processing a pipeline with concrete payload, we look for real data. The dataset obtained comes from the Metaspace 2020 website[8] and has 705MB – very small in comparison with other datasets. The same setup as above was used regarding the "serving side". First, executions of native Python 3.13 were made to observe its general profile, and later move to the sconified ones. It was observed that Metaspace has a varying range of runtime memory values to work out the payload: 512MB, 1024MB, 2048MB, and 4096MB. Metaspace sets these values when spawning new (`fork()`'ed) processes.

The enclave's memory has a perpetual rigid size that is never changed once it is created and it persists until the program exits; it has to be a number in powers of 2 to build the memory heap[9]. So, if the enclave has started with 1024MB, it will persist with this heap size until it exits, and it is shared between the program itself and TEE management activities; such that if the program has code identifying the total memory to work with, the enclave might eventually abort due to out of memory.

The SCONE runtime sets the enclave's memory using the environment variable "`SCONE_HEAP`" upon the program's startup. Both the TEE management activities and the size in powers of 2 restriction are taken into consideration to reserve the encrypted memory; therefore the SCONE runtime will "bump" the value set there to the next in power of 2. For example, to counter for the TEE management activities, if set to "`SCONE_HEAP=1500M`", the SCONE runtime will "bump" the enclave's encrypted memory size to the next value in power of 2, "*2048M*"; if set to "`SCONE_HEAP=2048M`", it will be bumped to "*4096M*"; if set to "`SCONE_HEAP=3G`", the enclave's memory will be bumped to "*4G*"; and so on.

Considering this TEE feature, together with the profiled varying runtime memory used by Metaspace, the sconified execution has to have the enclave's memory size set to cover both the maximum observed of 4096MB and the TEE management overhead. The value that passed the tests is "`SCONE_HEAP=7G`", which SCONE runtime will round up to 8GB.

Firstly, the execution is made on "*confidential computing simulation mode*", by setting the SCONE runtime with the environment variable "`SCONE_MODE=SIM`", where less restrictions are enforced on the enclave. Here is an excerpt of the pipeline execution with the selected dataset.

-
  ```
  $ SCONE_VERSION=1 SCONE_CONFIG_ID=horizon-mesh-20135-21162/lithops/mbolpayload\
  SCONE_MODE=SIM SCONE_HEAP=7G python3.13 /python/process-payload.py
  ...
  SCONE_MPROTECT=no
  SCONE_FORK=yes
  SCONE_FORK_OS=1
  SCONE_CONFIG_ID: horizon-mesh-20135-21162/lithops/mbolpayload
  musl version:  1.2.5
  SCONE version:  5.9.0-239-g941a371d1-dirty-sergei/lithops (base:  f9230ace) (2025-01-20
  ```

[8]AstraZeneca//Xenograft https://metaspace2020.org/dataset/2016-09-21_16h06m53s
[9]page 4: "*Table 2-2 (...) SIZE | Size of enclave in bytes; must be power of 2*"
https://www.intel.com/content/dam/develop/external/us/en/documents/329298-002-629101.pdf

```
14:09:47)
Enclave hash:  d5d66f2c6fac1e4f68d3199636889feb87f324ef1a899ec97ea55699eba3717d
2025-05-05 14:16:25,548 [INFO] config.py:139 - Lithops v3.6.1.dev0 - Python3.13
2025-05-05 14:16:25,548 [DEBUG] config.py:179 - Loading Serverless backend module:
singularity
2025-05-05 14:16:25,548 [INFO] config.py:139 - Lithops v3.6.1.dev0 - Python3.13
2025-05-05 14:16:25,548 [DEBUG] config.py:179 - Loading Serverless backend module:
singularity
2025-05-05 14:16:25,555 [DEBUG] config.py:219 - Loading Storage backend module:
minio
2025-05-05 14:16:25,562 [DEBUG] minio.py:38 - Creating MinIO client
2025-05-05 14:16:25,562 [DEBUG] minio.py:43 - Setting MinIO endpoint to
http://10.36.152.162:9000
...
2025-05-05 14:21:23,051 [INFO] invokers.py:186 - ExecutorID 7e88c3-0 | JobID M012
- Starting function invocation:  process_centr_segment() - Total:  32 activations
...
2025-05-05 14:21:33,542 [DEBUG] future.py:286 - ExecutorID 7e88c3-0 | JobID M012
- Got output from call 00023 - Activation ID: 5e202a2b440f
2025-05-05 14:21:33,625 [DEBUG] monitor.py:144 - ExecutorID 7e88c3-0 - Pending:
0 - Running:  0 - Done:  32
2025-05-05 14:21:33,657 [DEBUG] monitor.py:465 - ExecutorID 7e88c3-0 - Storage
job monitor finished
...
2025-05-05 14:22:01,295 [DEBUG] future.py:229 - ExecutorID 7e88c3-0 | JobID M013
- Got status from call 00000 - Activation ID: bc6931c690b9 - Time:  1.49 seconds
2025-05-05 14:22:01,298 [DEBUG] future.py:286 - ExecutorID 7e88c3-0 | JobID M013
- Got output from call 00000 - Activation ID: bc6931c690b9
...
2025-05-05 14:22:14,539 [DEBUG] executors.py:519 - ExecutorID 7e88c3-0 - Finished
getting results
```

Please note that at a later point in the pipeline execution, Metaspace had 32 activations, meaning it spawned 32 simultaneous processes running. This is very expensive in native execution and more expensive in a confidential computing one. This pipeline runs in confidential computing simulation mode, which means that it is less restrictive in the lowest level. For example, the size of the enclave is also set with `SCONE_HEAP`, and when a forked execution takes place, there is enough memory for everyone. Additionally, we can employ a fork optimization in simulation mode, making the execution faster than if it were using the actual TEE hardware.

Next, the execution is made on "*confidential computing hardware mode*", where the program will be handled inside the TEE hardware, and the SCONE runtime enforces more confidential computing restrictions. It is worth mentioning that, despite Metaspace being integrated with Lithops Singularity, it will eventually call the OS process `fork()` system call, to parallelize work. As mentioned above, every `fork()` is very expensive, due to the copying of the complete current enclave state to the new child process (and it is all encrypted, thus more processing activities).

Despite dividing the work among 6 Lithops Singularity instances, hosted in 3 powerful nodes (with CPU of 2.7GHz, 32 cores, 64GB of RAM), the same pipeline execution was very slow in comparison with the simulation version and consequently aborted. The investigation indicated that the EPC (Enclave Page Cache) was much smaller than the RAM size, so all processes will not fit in it when needed. EPC has been set to the new value of **16GB**, from the initial **1GB**. This helped mitigate the slowness, but the program still crashed:

-
  ```
  2025-04-23 18:49:16,815 [INFO] invokers.py:186 -- ExecutorID 9d1fa4-0 | JobID
  M013 - Starting function invocation: process_centr_segment() - Total: 32
  activations
  2025-04-23 18:49:16,815 [DEBUG] invokers.py:212 -- ExecutorID 9d1fa4-0 | JobID
  M013 - Worker processes: 6 - Chunksize: 6
  Traceback (most recent call last):
    File "/python/test-local.payload.py", line 48, in run_job
      job.run(debug_validate=True, perform_enrichment=False)
    File "/extra/sources/metaspace/sm/engine/annotation_lithops/annotation_job.py",
    line 228, in run
  ```

```
    ...
    File "/extra/sources/metaspace/sm/engine/annotation_lithops/annotate.py",
    line 309, in process_centr_segments
    File "/extra/sources/metaspace/sm/engine/annotation_lithops/executor.py",
    line 278, in map
      request_ids=[f.activation_id for f in futures],  # pylint: disable=not-an
      -iterable
  TypeError: 'NoneType' object is not iterable
```

The "*not iterable*" exception on the **futures** object indicates that the program failed to build it correctly due to running out of memory. On the Lithops Singularity server side, Python 3.13 processes aborted when the memory limit was reached. Example:

-
  ```
  root 1943778 1943765 5 08:09 ?  00:02:09 python3.13
  root 1974405 1943778 87 08:42 ?  00:04:07 python3.13
  root 1974577 1943778 23 08:42 ?  00:01:03 [python3.13] <defunct>
  root 1974685 1943778 26 08:42 ?  00:01:12 python3.13
  root 1974745 1943778 25 08:42 ?  00:01:08 python3.13
  ```

A successful execution in simulation mode spans to the 6 Lithops Singularity instances with on average 13 concurrent OS processes, with maximum of 15 per container. We tried to increase RAM on each of the 3 worker nodes to **256GB** and vary the setup of 6 Lithops Singularity instances to 3 approaching the maximum available for each node. Nevertheless, the same problems happened, but now reached a farther point in the pipeline, with 32 OS processes spawned simultaneously.

On a more refined analysis, the 32 OS processes, *i.e.* 32 enclaves with full copy state (of memory HEAP) from parent to child, would consume around the full 256GB if run on the same node; but regardless of that, they were split among the worker nodes and the memory limit was still an issue. Our best estimate now is to have **320GB** RAM per worker node with 3 Lithops Singularity instances. But the same problems occurred at the same point:

-
  ```
  2025-05-06 10:23:34,441 [INFO] invokers.py:186 -- ExecutorID b7c0b5-0 | JobID M011
  - Starting function invocation: merge_centr_df_segments() - Total: 32 activations
  2025-05-06 10:23:34,442 [DEBUG] invokers.py:212 -- ExecutorID b7c0b5-0 | JobID M011
  - Worker processes: 3 - Chunksize: 3
  Traceback (most recent call last):
    File "/python/process-payload.py", line 48, in run_job
      job.run(debug_validate=True, perform_enrichment=False)
    File "/extra/sources/metaspace/sm/engine/annotation_lithops/annotation_job.py",
    line 228, in run
    ...
    File "/extra/sources/metaspace/sm/engine/annotation_lithops/segment_centroids.py",
    line 205, in segment_centroids
      db_segms_cobjs = fexec.map_concat(
          merge_centr_df_segments, second_level_segms_cobjs, runtime_memory=512
      )
    File "/extra/sources/metaspace/sm/engine/annotation_lithops/executor.py",
    line 396, in map_concat
      results = self.map(func, args, runtime_memory=runtime_memory, **kwargs)
    File "/extra/sources/metaspace/sm/engine/annotation_lithops/executor.py",
    line 278, in map
      request_ids=[f.activation_id for f in futures],  # pylint: disable=not-an-
      iterable
  TypeError: 'NoneType' object is not iterable
  ```

Analyzing the executions with `strace` a peculiar behavior was identified: the forked processes had made a second *fork()*. Example:

-
  ```
  root 3374777 3374765 5 12:33 ?  00:05:01 python3.13
  root 3581635 3374777 16 13:54 ?  00:01:18 python3.13
  root 3581807 3374777 15 13:54 ?  00:01:14 [python3.13] <defunct>
  root 3582661 3374777 16 13:54 ?  00:01:15 python3.13
  ```

```
root 3585332 3581635 24 13:55 ?  00:01:38 python3.13
root 3586372 3582661 19 13:55 ?  00:01:17 python3.13
...
```

Taking 3 processes to analyze: the root parent PID 3374777 forks to 3581635, which subsequently will fork to 3585332. This means that considering only these 3 processes, there are "*8GB * 3 = 24GB*" of memory allocated, despite the application's runtime memory at that moment of 32 activations had been set by Metaspace being 512MB per activation. This happens because the root parent process started with the upper bound memory reservation of full **8GB** and less would crash the execution – and this cannot be changed throughout the program's lifetime. In sconified programs, each OS process is an enclave and when executed in hardware mode and a `fork()` is called the whole state is copied from parent to child.

This double-forking behavior indicates that it comes from internal decision of the Python interpreter (analyzing deeper its source code was outside of the scope of the project), as Lithops simply uses the standard core Python modules `subprocess` and `multiprocessing` without modifications; from the SCONE side, only the translation of system calls issued by the programs are made. Moreover, it is not possible to inspect inside the enclave (either with strace or any other tool) to understand precisely what is being done – it is a security feature by design.

The runtime memory variation inside the Metaspace pipeline is related to inner business logic, and currently no external settings are possible to change such as to guide the behavior at any given time. In conclusion, regardless of how many activities are decided by the application, the current Metaspace version will *fork* the execution to parallelize work and does not benefit from the current Lithops Singularity mechanism of using *threads* instead. And more hardware cannot be always the answer for speeding up a system.

However, Metaspace can be supported by SCONE, benefiting from attestation policies to transfer data securely along the network, as long as it is setup to work on simulation mode, hence not benefiting from the security features provided by the TEE hardware.

4.2.3 Scalability challenges in sconified Metaspace

This subsection summarizes the hardware requirements and compares with the scalability challenges faced to support the sconified Metaspace. The cluster supporting Metaspace initially consisted of 3 worker nodes configured with CPU Intel(R) Xeon(R) Platinum 8458P[10] of 2.7GHz with 32 cores and 64GB of RAM, and Intel SGX for TEE; they hosted 6 Lithops Singularity instances.

We progressively upgraded the nodes setup, according to the investigations of failed runs:

1. Enclave Page Cache - EPC increased from **1GB** to **16GB**;
2. Memory increased from **64GB** RAM to **256GB** RAM;
3. Memory increased from **256GB** RAM to **320GB** RAM.
4. Memory returned to **320GB** RAM to baseline **64GB** RAM;

CPU speed is automatically adjusted if necessary; this Intel model has an automatic upper bound of **3.8GHz**[11].

In conclusion, confidential computing has a known trade-off of performance for security, and often more hardware made available can do a good job; but in this scenario, more hardware exclusively is not the answer for adequate scalability of sconified Metaspace with Lithops. The dataset used has no much more than 700MB and there are other datasets much larger than it. This is a scenario of a very specific niche, where the amount of data is often huge and the corresponding hardware addressed to it is already potent, but considering the porting to confidential computing, it becomes prohibitive to increase resources 500 times the provisioned baseline. This presents a great opportunity to future works on the integration of Metaspace with the new Confidential Computing Lithops Singularity backend, adapting Metaspace to benefit of the threads mechanics and confidential computing in their full extent.

4.2.4 Keycloak

Both systems ported and non-ported to SCONE can benefit from confidential computing auxiliary services. Keycloak is an identity and access manager that is used as an outsourcing user management. In cases that not every person with access to a particular system is allowed to submit certain computations, Keycloak offers an authentication mechanism that provides additional information so that the recipient system can assert whether

[10]Intel CPU `https://www.intel.com/content/www/us/en/products/sku/231742/intel-xeon-platinum-8458p-processor-82-5m-cache-2-70-ghz/specifications.html`

[11]What Is Intel® Turbo Boost Technology? `https://www.intel.com/content/www/us/en/gaming/resources/turbo-boost.html`

a user can or cannot access certain resources (or submit computation, in our case). Keycloak has been setup in the KIO infrastructure and provides services via `keycloak.neardata.eu:30443`.

The configurations are made in *realms*, that contain *users*, *roles*, *clients*, *client scopes*,and other *authentication configurations*.

- **Realm**: Logic domain containing clients, users and their roles;
- **Client**: Logic entity corresponding to systems to which users will refer when logging in to Keycloak. A client is the gateway to services on other recipient systems;
- **Client scopes**: Client feature; albeit the client can access a service, some special resources might be restricted to certain scopes, giving the recipient system flexibility to control what can be done within its perimeter;
- **User**: Entity in the realm representing a person or another system;
- **Roles**: User feature representing a role in the recipient system.

Broadly speaking, the workflow is this: users in each realm will have roles and will access clients to deliver their information to the recipient systems. Roles are labels that the recipient system uses to allow and proceed, or deny and abort the connection. Upon successful logging in, a data structure called "*access token*" is sent to the recipient system with details about the user, including their roles. The recipient system will certify the user's claims by requesting a *validation token* from Keycloak.

NEARDATA has 4 realms that cover the use cases: **genomics**, **transcriptonomics**, **metabolomics** and **surgomics**. Each realm has been setup with *users*, *clients* and *authentication configurations*. Considering that the majority of systems we have dealt in NEARDATA are developed in Python, to ease the way in to partners, Scontain is providing a Python module accessible via `pip`: **sconekc**[12].

Here is an example of how to use `sconekc` in an automated integration with secrets injection via attestation:

- install it with `pip install sconekc`.
- make a x509 client certificate
```
export KEYCLOAKCLICERTSUBJ="/C=ES/ST=CA/L=Barcelona/O=NEARDATA/OU=BSC/CN=genomics";
openssl req -x509 -newkey rsa:2048 -keyout key.pem -out cert.pem -sha256 -days 365 -nodes
-subj "$KEYCLOAKCLICERTSUBJ";
```
- With this Python code:
-
```
$ SCONE_CONFIG_ID=horizon-mesh-20135-21162/keycloak/kc-client-userpass \
python3 kc-client-injection.py
```

```python
import sconekc as kc

SERVER=os.environ['KEYCLOAK']
REALM=os.environ['MBOLREALM']
CLIENT=os.environ['MBOLCLIENT']
USERNAME=os.environ['MBOLUSER']
PASSWORD=os.environ['MBOLPASS']

try:
    AT=kc.get_access_token(SERVER, REALM, CLIENT, USERNAME, PASSWORD,
        cert='cert.pem', key='key.pem')
    print("Access Token")
    kc.showJWT(AT)
    VT=kc.get_validation_token(SERVER, REALM, AT, cert='cert.pem', key='key.pem')
    print("Validation Token")
    print(VT)
except Exception as e:
    print(f'Exception: {e}')
```

This should give something like:

[12]sconekc – Module to interact with Keycloak access token and validation token https://pypi.org/project/sconekc/

```
Access Token
Header:  {'alg': 'RS256', 'kid': 'Qt5SlaqaZtwfZKzc673xBLdOTn0vU72eht2JnyHAh-A',
 'typ': 'JWT'}
Token:   {'azp': 'serverless-genomics',
 'email': 'frogins@neardata.eu',
 'exp': 1758021156,
 'iat': 1758018816,
 'iss': 'https://keycloak.neardata.eu:30443/realms/genomics',
 'name': 'Frobor Laggins',
 'preferred_username': 'frogins',
 'realm_access': {'roles': ['trypanosome',
                            'bos-taurus',
                            'human',
                            'default-roles-genomics']},
 'sid': 'e5854207-e800-495a-aa19-1b7a01d069da',
 'sub': 'd58e846b-8e79-4d0b-9a5a-c4be0d17f530',
 'typ': 'Bearer'}
Signature:   'ZHQVWnTkcJohatnWnXSPu_HYQuKpGb0K_PK1K-6J5dADo...Xg6hGx_2shdrFV9YLEjDtGQ'
Issued at:  2025-09-16 12:33:36 (localtime)
Not before: Undefined (localtime)
Expiration: 2025-09-16 13:12:36 (localtime)

Validation Token
{'sub': 'd58e846b-8e79-4d0b-9a5a-c4be0d17f530', 'preferred_username': 'frogins', 'email':
'frogins@neardata.eu', 'accesstokentimestampissuing': 1758018816, 'accesstokendatetime\
issuing': 'Issued at:  2025-09-16 12:33:36 (localtime)', 'accesstokentimestampexpiring':
1758021156, 'accesstokendatetimeexpiring': 'Expiration: 2025-09-16 13:12:36 (localtime)',
'validationtokentimestampvalidation': 1758018816, 'validationtokendatetimevalidation':
'Validated on: 2025-09-16 12:33:36 (localtime)'}
```

The code above is the trivial workflow using Keycloak's REST API, where a user is authenticated with their password via secure HTTPS channel. The communication is made on behalf of the "genomics" realm user "frogins", via the client "serverless-genomics" that started the communication requiring services from a third party (the recipient system). Upon authentication, the *access token* is sent to the recipient system to check which roles have been assigned to the user. This allows for a **RBAC – Role Based Access Control** filtering. And as important as getting the access token from the user, is the validation token requesting by the recipient system. The *validation token* is generated by the same identity and access manager that generated the access token; therefore, the same access token is sent to the server for validation; and, once issued, the recipient can assert that everything is trustworthy.

The following snippet is a more refined approach of authentication that does not mediate the passing of username and password to Keycloak, but instead uses the Keycloak login page itself. This is an interactive integration approach, with injection of secrets via attestation.

- `$ SCONE_CONFIG_ID=horizon-mesh-20135-21162/keycloak/kc-loginpage-lt-multiproc \`
  `python3 kc-multiprocessinglithops.py`

```python
AUTH_SERVER = os.environ['KEYCLOAK']
REALM = os.environ['MBOLREALM']
CLIENT_ID = os.environ['MBOLCLIENT']
ROLE = os.environ['MBOLROLE']
REDIRECT_URI = "http://127.0.0.1:4443/callback"
BROWSER = "/usr/bin/opera"
BROWSER_PARAMS = "--private"

server = threading.Thread
kcsession = kc.KeycloakSession()
browser_t = threading.Thread

if __name__ == "__main__":
    start_server()

    x=lambda: kc.login_browser_auth(kcsession, AUTH_SERVER, REALM, CLIENT_ID,
        REDIRECT_URI, BROWSER, BROWSER_PARAMS)
    t=threading.Thread(target=x)
    t.start()
    t.join()
...
    print(f"Performing verification: RBAC access={ROLE}")
    access_token_json=json.loads(kcsession.access_token_json)
    header, access_token, signature, issued, started, validto = kc.sliceJWT(
        access_token_json["access_token"])
    if ROLE in access_token['realm_access']['roles']:
        print("User IS AUTHORIZED to proceed with computation")
        header, payload, signature, issued, started, validto = kc.sliceJWT(
            access_token_json["access_token"])
...
        VT=kc.get_validation_token(AUTH_SERVER, REALM,
            access_token_json['access_token'])
        print(f"validation_token={VT}")
        print(f"validation_token.name={VT['name']}")
        if VT['name'] != "":
            from lithops.multiprocessing import Pool
            def double(i):
                return i * 2
            with Pool() as pool:
                result = pool.map(double, [1, 2, 3, 4])
                print(result)
    else:
        print("User IS FORBIDEN to proceed with computation")
    kc.logout_session(kcsession, AUTH_SERVER, REALM)
```

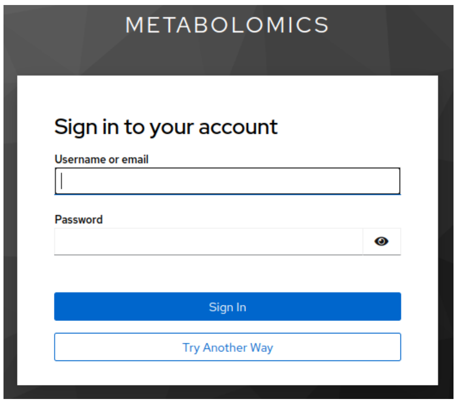
This mechanics uses the Keycloak's endpoint `/auth` (used with browser interaction) instead of `/token` (which is used by the trivial access token generation). The endpoint `/auth` requires that a callback URI (open port or HTTP URL) is prepared to receive a response code, which will then be sent to Keycloak to generate the access token on behalf of the user. Please note that in this approach both the starting program the user is interacting with and the recipient system have no access to their password; upon authentication on the Keycloak webpage, the recipient system will use the authorization code received and submit it to the same `/token` and get the same access token with the user's details, roles, etc.

The module `sconekc` provides the function **login_browser_auth()** that will build the configurations to submit to `/auth` and will open the default desktop's browser or use the one specified by the variable **BROWSER**. It is essential to define the correct `redirect URI`. If is different from what is configured and allowed for that `client`, the logging in fails and no authorization code is generated. The redirect URI has to be accessible by the user resources or the recipient system.

The function **callback_endpoint()** is used to receive the authentication token generated from the Keycloak's login webpage and will request the access token with the authorization code received. It has to have setup the same redirect URI that was used initially. It is not an HTTP server; instead, it is a function that can be used, for instance, in a Flask handler deployment. And this is what has been done in the application above: despite being a command line Python program, it will load a temporary web server exclusively to listen for the callback communication and proceed to obtain the access token. The program above brings the integration of Keycloak with Lithops, demonstrating that the confidential computing covers the broadest extent of the computation continuum.

If the integration of `sconekc` is made with the traditional username+password mediation, the best option is to use environment variables inject via SCONE attestation (e.g. `os.environ['MBOLPASS']`), and this is the option-of-choice if the pipeline non-interactive. Otherwise, the authentication via browser, for interactive execution is the way to go. An important authentication configuration introduced is the "*passkeys*" support in Keycloak. Once the authentication is made using the browser, the use of passkeys can be employed and substitute the use of passwords.

The example above will allow access to users from the realm "*metabolomics*", accordingly to the successful login via the webpage. Once the Keycloak login page is open, the user can choose to login by typing their username and password or use the their passkey issued in advance by Keycloak [6]. It has been setup to allow both username + password typing and the passkeys passing, but it can be setup to exclusively passkeys.


METABOLOMICS

Sign in to your account

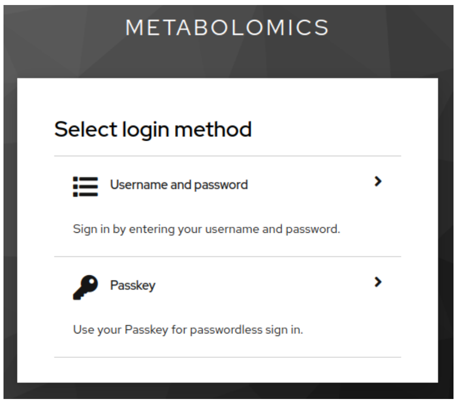
Username or email

Password

Sign In

Try Another Way

Keycloak login webpage.


METABOLOMICS

Select login method

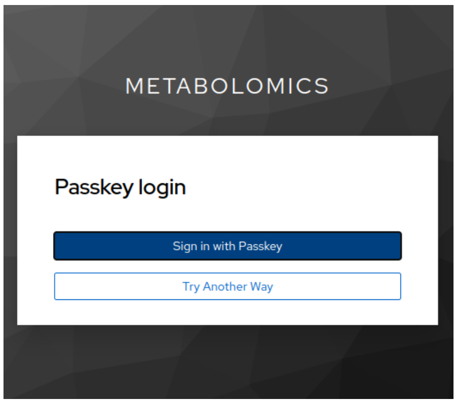
Username and password

Sign in by entering your username and password.

Passkey

Use your Passkey for passwordless sign in.

Login method selection.

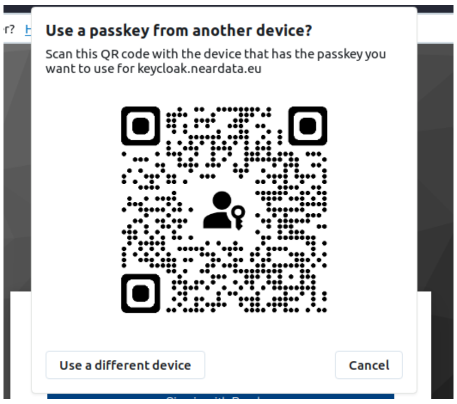

METABOLOMICS

Passkey login

Sign in with Passkey

Try Another Way

Select passkeys method.

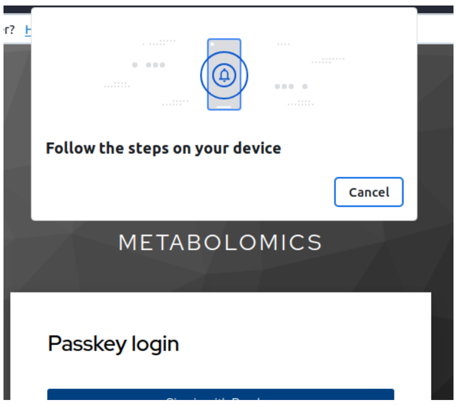

Use a passkey from another device?

Scan this QR code with the device that has the passkey you want to use for keycloak.neardata.eu

Use a different device

Cancel

QR-CODE to read from mobile.

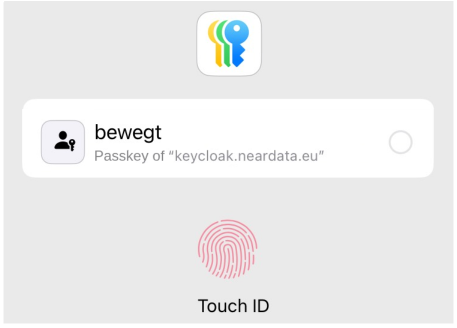

Follow the steps on your device

Cancel

METABOLOMICS

Passkey login

Continue from mobile.


bewegt

Passkey of "keycloak.neardata.eu"

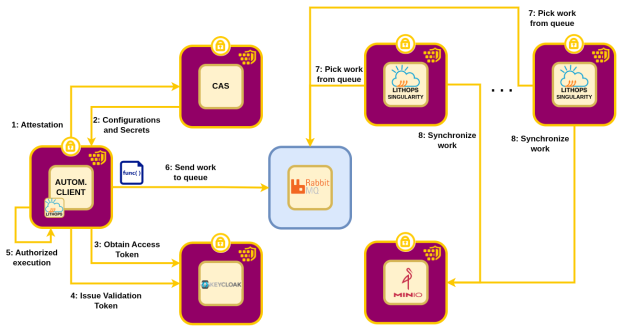
Touch ID

Allow to use passkey from mobile.

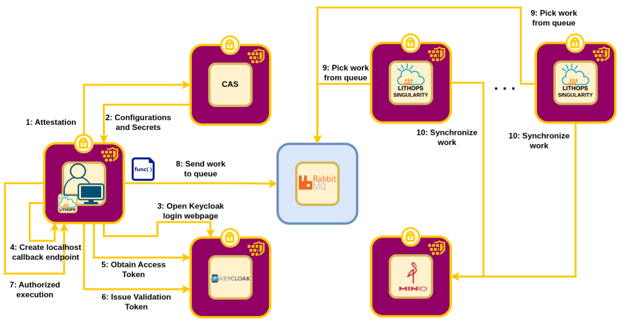
Table 6: Keycloak authentication webpage using passkeys

The diagrams [3] depict both automated and interactive integrations of user authentication.

Keycloak configuration : here are the steps to setup the Keycloak server installed by the confidential mesh of services[4.2.5]. After installation, the temporary bootstrap administrator user + password have to be "sanitized" (new permanent user created and given the administrator rights, plus disabling or deleting the bootstrap administrator user). This step is done in the web management interface of **master** realm. The complete list of configurations can be found in table [7].



(a) Keycloak automated client integration.



(b) Keycloak interative client via browser integration.

Figure 3: Keycloak integration methods.

1. Go to: menu |Users| » [Add user] » Fill in: "username" and "email" » Switch: "Email verified" to **On** » [Create];
2. Go to: menu |Credentials| » [Credential Reset] » [Set password] » "*Enter the new administrator's password*" » Switch: "Temporary" to **Off** » [Save];
3. Go to: menu |Role mapping| » [Assign role] » Switch: "Filter by realm roles" » Select: **admin** » [Assign];
4. Go to: menu |Users| » select the bootstrap **admin** user » Switch: "Enabled" to **Off**;
5. Logout and login with the new administrative user.

Now the realms, clients, roles, and users can be set up.

1. Go to: drop down menu "**master**" » [Create real] » Enter: "Realm name" » [Create]
   The interface changes to the new realm. If not, select it from the drop down menu.
2. Go to: menu |Client scopes| » [Create client scope] » Fill in: "Name" with "**openid**" » Switch: "Display on consent screen" **Off** » [Save];
   The scope "**openid**" is mandatory and is used to properly verify an authenticated user by a recipient system (i.e. obtain a validation token);
   If necessary, create other scopes;
3. Go to: menu |Clients| » [Create client] » Fill in: "Client ID" » [Next] » [Next] » Fill in: "Valid redirect URIs" with "**http://127.0.0.1:4443/callback**" » [Save] ;
   Enter as many redirect URIs you need in the client or as many new clients; The interface changes to the new client.
4. Go to: menu |Client scopes| » [Add client scopes] » Select: "**openid**" and any other client scope needed » [Add-Default];
   When clicking [Add] select "Default" to have those scopes always present in that client when the user requests access to it.
5. Go to: menu |Realm roles| » [Create role] » Fill in: "Role name" » [Save];
6. Go to: menu |Users| » [Add user] » Fill in: "username" and "email", and "First name" and "Last name" » Switch: "Email verified" to **On** » [Create];
7. Go to: menu |Credentials| » [Credential Reset] » [Set password] » Switch: "Temporary" to **Off** » [Save];
8. Go to: menu |Role mapping| » [Assign role] » Switch: "Filter by realm roles" » Select: the roles created before » [Assign];

Realm	Clients	Client scopes	Roles	Users
genomics	serverless-genomics	openid	human	frogins
		variant-calling	trypanosome	sanfwise

metabolomics	lithops-backend	openid	lithops_k8s	allota
		sm-az-rat-brains	lithops_singularity	bewegt
		sm-ct26-xenograf		myuser
surgomics	pravega-client	openid	pravega_kidney	haneks
		surg-ongo	pravega_liver	kellsen
			pravega_lungs	
transcriptonomics	atlas-terraform	openid	atlas_salmon	mofarrej
		infra_ec2	atlas_star	tivoli
		infra_hpc		

Table 7: Summary of the Keycloak settings

4.2.5 Confidential Mesh of Services

A Kubernetes cluster with nodes that support TEE hardware can host a mesh of confidential services that will communicate with authorized parties and are controlled by the attestation system **CAS**.

A CAS instance, with replicas spread among the cluster's nodes, is controlled by the systems administrator using the program "**SCONE Operator**"[13]. The Operator is a Kubernetes plugin for `kubectl` installed[14] in the systems administrator's workstation and is capable of provisioning and managing CAS installations in the cluster. Here is an example of installing a new CAS in the cluster:

- `kubectl provision neardata-cas cas --dcap-api bade60cc793457139e71d472420c3ea3`

The DCAP API Key above has to be the same used to install the Operator. You can get a DCAP API Key from the Intel website[15].

The mesh of services is managed by the application "**SCONECTL**". SCONECTL helps to transform cloud-native applications into cloud-confidential applications. It supports converting native services into confidential services and services meshes into confidential services meshes.

The configuration of a mesh of services, "`mesh.yaml`" transcribed below, is made through a YAML manifest file. Here is the confidential mesh of services containing Lithops, MinIO, and Keycloak with MariaDB:

```yaml
apiVersion: scone/5.8
kind: mesh

cas:
  - name: neardata-cas-v1 # cas used to store the policy of this application
    alias: ["image", "security", "access", "attestation"]
    cas_url: neardata-cas-v1.default
    cas_key: 4REHj3wFQRt6yi5b3c5BDnRLLxyuotTq9y1FDbz9FRNDb32kxQ
    tolerance: "--accept-configuration-needed --accept-group-out-of-date
--accept-sw-hardening-needed"
    mode: SignedManifest
    cas_encryption_key: 3704f3e07fafb5961259f12e9c751bb8419b9b90067afa65acd028c9992db

policy:
  namespace: horizon-mesh-20135-21162
  tolerate: debug-mode
  ignore_advisories: '*'
  attestation_policy_name: null
  access_policy_name: null
  security_policy_name: null
  image_policy_namespace: null
  maa: null

# Define environment variables
# These variables will be used to dynamically configure the policies and services
```

[13]SCONE Kubernetes Operator https://sconedocs.github.io/1_scone_operator/
[14]Deploying & Reconciling the SCONE Operator https://sconedocs.github.io/2_operator_installation/
[15]Intel® Provisioning Certification Service https://api.portal.trustedservices.intel.com/provisioning-certification

```
# - we define two special services:
#   - "global": these definitions are shared by all services
#   - "helm": this behaves like the "global" service - the intention is to
#             collect helm specific definitions here
# - *_host: container's hostname
# - *_port: service port
# - *_fqdn: service hostname accessible elsewhere
# - *_service_name: service name in the policies
# - cluster_namespace: Kubernetes namespace of installation
# - pvc_identification: disambiguation PVC identification
# - cluster_release: Pods/Depoyments/Services name prefix
# - domain_name: DNS domain name
###
# - and for each individual service defined in the services section
#   - these definitions are only visible for this service
#   - these definitions overwrite any definitions of the "global" or "helm" section
env:
  - service: global
    env:
      - name: imagePullSecrets
        value: sconeapps
      - name: useSGXDevPlugin
        value: scone
      - name: sgxEpcMem
        value: 128

      - name: database_host
        value: horizon-mesh-mariadb
      - name: database_port
        value: 3306

      - name: keycloak_host
        value: horizon-mesh-keycloak
      - name: keycloak_fqdn
        value: keycloak.neardata.eu
      - name: keycloak_service_name
        value: keycloak

      - name: minio_host
        value: horizon-mesh-minio
      - name: minio_fqdn
        value: minio.neardata.eu
      - name: minio_service_name
        value: minio

      - name: lithops_host
        value: horizon-mesh-lithops
      - name: lithops_fqdn
        value: lithops.neardata.eu
      - name: lithops_service_name
        value: lithops

      - name: cluster_namespace
        value: default
      - name: pvc_identification
        value: horizon-storage
      - name: cluster_release
        value: horizon-mesh
      - name: domain_name
        value: neardata.eu

      - name: pkcs_generator_image # Python image with libraries for the Keycloak
 # initialization container used to generate PKCS12 credentials
        value: registry.scontain.com/sconecuratedimages/experimental/lithops:3.6.1-
        metaspace-5.9.0-239-g941a371d1


  ###
  # database supporting Keycloak
  # - sqlInitScript: injected via attestation; SQL script to create user & database
  #                  by the first load/configuration
  # - etcMyCnf: injected via attestation; /etc/my.cnf MariaDB configuration file
  - service: mariadb
    env:
      - name: sqlInitScript
        value: |
         |
                  GRANT ALL PRIVILEGES ON *.* TO 'root'@'%' IDENTIFIED BY '$$SCONE::
         mariadbrootpassword$$';
                  --
                  CREATE DATABASE keycloak;
                  --
                  CREATE USER 'ciam'@'%' IDENTIFIED BY '$$SCONE::keycloakdbpassword
         $$';
                  GRANT ALL PRIVILEGES ON *.* TO 'ciam'@'%' REQUIRE X509 WITH GRANT
         OPTION;
                  FLUSH PRIVILEGES;
                  SHOW GRANTS FOR 'ciam'@'%';

      - name: etcMyCnf
        value: |
         |
                  # This group is read both both by the client and the server
                  # use it for options that affect everything
                  [client-server]
                  # This group is read by the server
                  [mysqld]
                  user=mysql
                  skip-host-cache
                  skip-name-resolve
                  # Disabling symbolic-links is recommended to prevent assorted
         security risks
                  symbolic-links=0
                  # Network
                  bind-address = 0.0.0.0
                  port = 3306
                  # Encryption parameters
                  plugin_load_add = file_key_management
                  file_key_management_filename = /etc/keys.txt
                  file_key_management_encryption_algorithm = aes_cbc
                  encrypt_binlog = 1
                  innodb_encrypt_tables = ON
                  innodb_encrypt_log = ON
                  innodb_encryption_threads = 4
                  innodb_encryption_rotate_key_age = 0 # Do not rotate key
                  innodb-tablespaces-encryption = ON
```

```
                    encrypt_tmp_files = ON
                    innodb_log_group_home_dir = /external
                    innodb_data_home_dir = /external
                    ## Trying direct user access
                    # Enabling TLS for MariaDB server
                    ssl
                    ssl_cert = /etc/server.crt
                    ssl_key = /etc/server.key
                    ssl_ca = /etc/mariadb-ca.crt
                    ## Enabling TLS for MariaDB clients
                    [client]
                    ssl_cert = /etc/client.crt
                    ssl_key = /etc/client.key
                    ssl-verify-server-cert


  ###
  # * Keycloak is named 'ciam' (Confidential Identity and Access Manager)
  # - imagePython:            - Python image for startup container to generate
  #                             database PKCS12 client credentials
  #                             generator
  # - suffixPkcsSession:      - used to format the auxiliary session name containing
  #                             the database PKCS12 client credentials. Example:
  #                             {{the_same_namespace}}-aux-2
  - service: keycloak
    env:
      - name: sconeHashJavaKeycloak
        value: 597a4b1bf92283b93b423c44b9aba30bebbf3cbf456a2a6459832355631f1beb
      - name: sconeKeycloakImage
        value: registry.scontain.com/sconecuratedimages/experimental/keycloak:26-
        alpine3.20-sconectl-5.9.0-rc.5
      - name: sconeKeycloakImageUnderline
        value: registry_scontain_com_sconecuratedimages_experimental_keycloak_26_
        alpine3_20_sconectl_5_9_0_rc_5
      - name: databaseConnectUrl            # env var KC_DB_URL
        value: jdbc:mariadb://horizon-mesh-mariadb/keycloak?user=ciam&
        password=$$SCONE::keycloakdbpassword$$&sslMode=trust&
        keyStore=/tls/mariadb.pkcs12&
        keyStorePassword=$$SCONE::MariadbPkcs12Password$$
      - name: pathJavaKc                    # env var PATH
        value: "/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:
        /usr/lib/jvm/java-21-openjdk/jre/bin:/usr/lib/jvm/java-21-openjdk/bin:
        /keycloak/bin"
      - name: ldLibraryPathJavaKc           # env var LD_LIBRARY_PATH
        value: /usr/lib/jvm/java-21-openjdk/lib/server:/usr/lib/jvm/java-21-openjdk/
        lib:/usr/lib/jvm/java-21-openjdk/../lib
      - name: javaHomeJavaKc                # env var JAVA_HOME
        value: /usr/lib/jvm/java-21-openjdk
      - name: kcAdditionalParams
        value: --features=preview --features=passkeys

      - name: javaXms                       # JVM minimal heap: -Xms
        value: 256m
      - name: javaXmx                       # JVM maxmimum heap: -Xmx
        value: 2048m
      - name: javaXMetaspace      # classes metadata initial size: -XX:MetaspaceSize
        value: 96M
      - name: javaXMaxMetaspace # classes metadata maximum size: -XX:MaxMetaspaceSize
```

```
        value: 256m

      - name: kcSconeEdmm                   # env var SCONE_EDMM_MODE
        value: auto
      - name: kcSconeMinHeap                # env var SCONE_MIN_HEAP
        value: 50m
      - name: kcSconeHeap                   # env var SCONE_HEAP
        value: 12G
      - name: kcSconeStack                  # env var SCONE_STACK
        value: 8M
      - name: kcSconeLog                    # env var SCONE_LOG
        value: ERROR
      - name: kcSconeNetshield              # env var SCONE_NETWORK_SHIELD
        value: protected

      # TEMPORARY application admin user + password used to bootstrap initial setup
      # MUST HAVE a new admin user added via web management interface and this
      # removed right after installation
      - name: kcAdminUser                   # env var KC_BOOTSTRAP_ADMIN_USERNAME
        value: temporarybootstrapadmin
      - name: kcAdminPassword               # env var KC_BOOTSTRAP_ADMIN_PASSWORD
        value: $$SCONE::keycloakbootstrappassword$$
      - name: kcDbManager                   # env var KC_DB
        value: mariadb

      - name: httpPort                     # env var KC_HTTP_PORT. localhost service
        value: 30080
      - name: httpsPort                     # env var KC_HTTPS_PORT. Public service
        value: 30443

      - name: imagePython
        value: registry.scontain.com/sconecuratedimages/experimental/python:3.10-
        pkcsandcas-5.9.0-rc.11
      - name: suffixPkcsSession
        value: aux-2

  ###
  # cloud object storage
  # MinIO environment variables have default values, but need to be setup by design
  - service: minio
    env:
      - name: sconeHashMinio
        value: 1d9e2c61a12e2997d008238ee106824650e6c8e20ee24d053fdab68cb0109600
      - name: sconeMinioImage
        value: registry.scontain.com/sconecuratedimages/experimental/minio:master-
        alpine3.21-sconectl-5.9.0-831-g2f59cb75a
      - name: sconeMinioImageUnderline
        value: registry_scontain_com_sconecuratedimages_experimental_minio_master_
        alpine3_21_sconectl_5_9_0_831_g2f59cb75a

      - name: mnSconeHeap               # env var SCONE_HEAP
        value: 7G
      - name: mnSconeAllowDlOpen        # env var SCONE_ALLOW_DLOPEN
        value: 1
      - name: mnSconeLog                # env var #SCONE_LOG
        value: ERROR
      - name: mnSconeNetshield          # env var SCONE_NETWORK_SHIELD
        value: protected

      - name: mnRootUser                # env var MINIO_ROOT_USER
        value: minioroot
      - name: mnRootPassword            # env var MINIO_ROOT_PASSWORD
        value: $$SCONE::minioadminpassword$$
      - name: mnRootUserFile            # env var MINIO_ROOT_USER_FILE
        value: access_key
      - name: mnRootPasswordFile        # env var MINIO_ROOT_PASSWORD_FILE
        value: secret_key
      - name: mnDataDirVolume           # path to mounted volume
        value: /minio-data
      - name: mnMcConfigDir             # env var MC_CONFIG_DIR
        value: /minio-data/.mc
      - name: mnAccessKeyFile           # env var MINIO_ACCESS_KEY_FILE
        value: access_key
      - name: mnSecretKeyFile           # env var MINIO_SECRET_KEY_FILE
        value: secret_key
      - name: mnKmsSecretKeyFile        # env var MINIO_KMS_SECRET_KEY_FILE
        value: kms_master_key
      - name: mnConfigEnvFile           # env var  MINIO_CONFIG_ENV_FILE
        value: config.env
      - name: mnServicePort             # Kubernetes service settings
        value: 9000
      - name: mnConsolePort             # Kubernetes service settings
        value: 9001
      - name: mnServiceAddress          # value for parameter --address
        value: :9000
      - name: mnConsoleAddress          # value for parameter --console-address
        value: :9001

  ###
  # Lithops singularity
  - service: lithops
    env:
      - name: sconeHashPythonLithops
        value: b5dc3e21b285878826efa64c34ede05c8ae93684c5397698b7701d1c14de10cc
      - name: sconeLithopsImage
        value: registry.scontain.com/sconecuratedimages/experimental/lithops:3.6.1-
        metaspace-5.9.0-239-g941a371d1
      - name: sconeLithopsImageUnderline
        value: registry_scontain_com_sconecuratedimages_experimental_lithops_3_6_1_
        metaspace_5_9_0_239_g941a371d1
      - name: pathLithops
        value: "/usr/local/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:
        /sbin:/bin"

      - name: ltSconeEdmm               # env var SCONE_EDMM_MODE
        value: auto
      - name: ltSconeMinHeap            # env var SCONE_MIN_HEAP
        value: 50m
      - name: ltSconeHeap               # env var SCONE_HEAP
        value: 7G
      - name: ltSconeAllowDlOpen        # env var SCONE_ALLOW_DLOPEN
        value: 1
      - name: ltSconeLog                # env var #SCONE_LOG
        value: ERROR
      - name: ltSconeMode               # env var SCONE_MODE
        value: SIM
      - name: ltSconeFork               # env var SCONE_FORK
        value: 1
      - name: ltSconeForkOs             # env var SCONE_FORK_OS
        value: 1
      - name: ltSconeSysLibs            # env var SCONE_SYSLIBS
        value: 1
      - name: ltSconeNetshield          # env var SCONE_NETWORK_SHIELD
        value: protected

      - name: ltPythonGil               # env var PYTHON_GIL
        value: 0
      - name: ltKubeReplicas            # YAML: lithops_server: replicaCount:
        value: 6
      - name: ltKubeRequestsMem         # YAML: resources: requests: memory:
        value: 16Gi
      - name: ltKubeRequestsCpu         # YAML: resources: requests: cpu:
        value: 12
      - name: ltKubeLimitsMem           # YAML: resources: limits: memory:
        value: 24Gi
      - name: ltKubeLimitsCpu           # YAML: resources: limits: cpu:
        value: 20
      - name: rabbitmqUser              # command line argument in URL amqp://
        value: $$SCONE::rabbitmquser$$
      - name: rabbitmqPassword          # command line argument in URL amqp://
        value: $$SCONE::rabbitmqpassword$$
      - name: rabbitmqServerPort        # command line argument in URL amqp://
        value: 10.36.152.162
      - name: rabbitmqUrl               # command line argument in URL amqp://
        value: test

services:
  - name: keycloak
    image: registry.scontain.com/sconecuratedimages/experimental/keycloak:26-
    alpine3.20-sconectl-5.9.0-rc.5
  - name: mariadb
    image: registry.scontain.com/sconecuratedimages/experimental/mariadb:10.4.24-
    sconectl-keycloak-5.8.0
  - name: minio
    image: registry.scontain.com/sconecuratedimages/experimental/minio:master-
    alpine3.21-sconectl-5.9.0-831-g2f59cb75a
  - name: lithops
    image: registry.scontain.com/sconecuratedimages/experimental/lithops:3.6.1-
    metaspace-5.9.0-239-g941a371d1

helm_extra_values:
  mariadb:
    persistence:
      enabled: true
      existingClaim: kc-horizon-storage-maria-data-pvc
      size: 8Gi
      mountPath: /var/lib/mysql
      accessModes:
        - ReadWriteOnce
    extraVolumes:
      - name: external
        persistentVolumeClaim:
          claimName: kc-horizon-storage-maria-external-pvc
      - name: vartmp
```

```
          persistentVolumeClaim:
            claimName: kc-horizon-storage-maria-vartmp-pvc
      extraVolumeMounts:
        - name: external
          mountPath: /external
        - name: vartmp
          mountPath: /var/tmp
```

The confidential mesh of sevices is installed using/storing CAS configurations in the systems administrator home directory:

- `sconectl apply --cas-config $HOME/.cas -f mesh.yaml`

The mesh YAML manifest is divided in sections:

- **cas**: contains information to reach the CAS server and configurations to use it, trust it based on the key at "`cas_key`", and the special feature "`mode`" which allows two values "`SignedManifest`" and "`Encrypted-Manifest`", that will upload manifests to the cluster either signed or encrypted and will be addressed properly with by the CAS;
- **policy**: defines policy related configurations. The feature "`namespace`" represents the session name that starts the chain of authorized services in a set of policies;
- **env**: environment variables used to setup policies and cluster manifests. They are set up by the parameter "`service`", that represents the services names that will be managed (e.g. `service: mariadb` etc.). A special "`service: global`" is used to setup variables used by SCONECTL core mechanisms and also by the services. Each variable is placed below the corresponding "`service: env:`" and defined by the parameter "`name`" followed by the parameter "*value*";
- **services**: a pair of parameters "`name`" and "*image*" telling SCONECTL the service name and its corresponding Docker image;
- **helm_extra_values**: consists of paths to values that will be used on cluster manifest files "`values.yaml.template`" of each service, used by `Helm` to generate Kubernetes manifests. Example inside the file: "`claimName: {{ .Values.persistence.existingClaim }}`".

The mesh of services[4] installed will have 6 instances of Lithops Singularity, one MinIO serving the system and the users, one Keycloak + MariaDB identity and access manager supporting the granularity of access at the levels of authenticated users and their roles. Data at rest, in transit, and in use are protected by the SCONE mechanisms of encryption, authentication, and attestation.

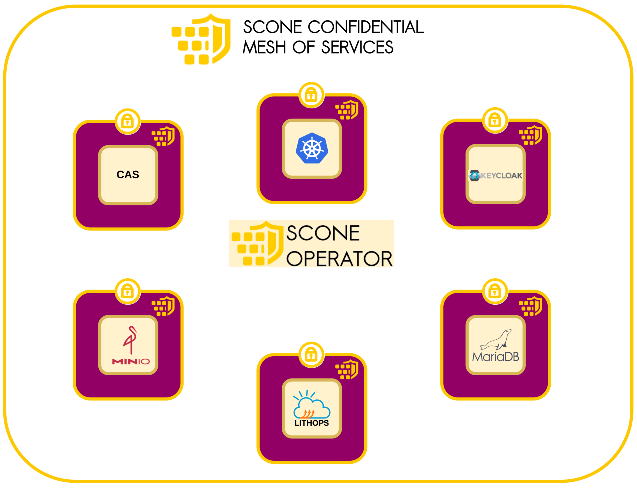


Figure 4: Confidential Mesh of Services.

The policies used by the services policies have special syntax placeholders (enveloped between double curly-brackets, e.g. "**{{someplaceholdername}}**") that are controlled by SCONECTL to substitute at configuration and upload time to CAS. Example from the "`lithops.yaml`" service policy manifest: `PYTHON_GIL:`

"{{ltPythonGil}}", obtained from the `mesh.yaml`, at "env: » service: lithops » env: » - name: ltPythonGil »
This will setup the environment variable "`PYTHON_GIL=0`" in the policy file that will be made available to the program upon attestation.

The policies and cluster manifests settings are embedded in special Docker images prepared for SCONECTL compatibility. Refer to "`mesh.yaml`" [transcription above] to learn which environment variables the settings go to. To ease the reader's understanding, the policies presented below highlight the most important parts and show with "`...`" the repeated configurations among them (like the shared configuration below) and the environment variables that come from "`mesh.yaml`".

Shared configuration for services policies: All policies share the same set of configurations below. Access, security and attestation are managed by SCONECTL:

```
access_policy: $$SCONE::access_policy$$

creator:
  {{session_creator}}

security: $$SCONE::security_policy$$
...
    attestation:
      - $$SCONE::attestation_service$$
...
secrets:
  - name: access_policy
    import:
      session: {{access_policy_name}}
      secret: access_policy
      cas_url: {{cas.access.cas_url}}
      cas_key: {{cas.access.cas_key}}
  - name: security_policy
    import:
      session: {{security_policy_name}}
      secret: security_policy
      cas_url: {{cas.security.cas_url}}
      cas_key: {{cas.security.cas_key}}
  - name: SCONE_SESSION_ID_
    kind: ascii
    value: "{{RANDOM}}"
  - name: attestation_service
    import:
      session: {{image_policy_name}}
      secret: mrsigner_mrenclave_versions
      cas_url: {{cas.image.cas_url}}
      cas_key: {{cas.image.cas_key}}
```

Lithops service policies: It is started from the Singularity `entry_point.py`. It has one parameter pointing to the RabbitMQ URL, that is filled in with SCONECTL controlled variables and provisioned to the program upon attestation; it will be hidden from the Operating System when a "`ps -ef`" is executed, for example, it will only show the program "`python3.13`".

```
services:
  # server policy
  - name: lithops-singularity
    command: python3.13 entry_point.py "amqp://{{rabbitmqUser}}:{{rabbitmqPassword}}
      @{{rabbitmqServerPort}}/{{rabbitmqUrl}}"
    pwd: /app
    environment: # generated from service manifest
      #####
      # application configuration
...
```

```
      SCONE_NETWORK_SHIELD: {{ltSconeNetshield}}
      SCONE_NETWORK_SHIELD_SERVER_1: {{ltSconeNetshield}}
      SCONE_NETWORK_SHIELD_SERVER_1_PORT: TCP:8080
      SCONE_NETWORK_SHIELD_SERVER_1_CLIENT_AUTH: $$SCONE::ca_certificate$$
      SCONE_NETWORK_SHIELD_SERVER_1_IDENTITY: $$SCONE::server_certificate:privatekey
        :pkcs8:pem$$$$SCONE::server_certificate:crt:pem$$
      SCONE_NETWORK_SHIELD_CLIENT_1: {{ltSconeNetshield}}
      SCONE_NETWORK_SHIELD_CLIENT_1_DESTINATION: TCP:confidential:9000
      SCONE_NETWORK_SHIELD_CLIENT_1_DESTINATION_IP: "*"
      SCONE_NETWORK_SHIELD_CLIENT_1_SERVER_AUTH: $$SCONE::ca_certificate$$
      SCONE_NETWORK_SHIELD_CLIENT_1_IDENTITY: $$SCONE::client_certificate:privatekey
        :pkcs8:pem$$$$SCONE::client_certificate:crt:pem$$
...
  - name: hello
    command: python3.13 hellolithops.py
    pwd: /python
    environment: # generated from service manifest
      SCONE_NETWORK_SHIELD: {{ltSconeNetshield}}
...
  - name: multiproc
    command: python3.13 multiprocessinglithops.py
    pwd: /python
    environment: # generated from service manifest
      SCONE_NETWORK_SHIELD: {{ltSconeNetshield}}
...
  - name: mboltest
    command: python3.13 test-local.py
    pwd: /python
    environment: # generated from service manifest
      SCONE_NETWORK_SHIELD: {{ltSconeNetshield}}
...
  - name: mbolpayload
    command: python3.13 process-payload.py
    pwd: /python
    environment: # generated from service manifest
      SCONE_NETWORK_SHIELD: {{ltSconeNetshield}}
...
    attestation:
      - $$SCONE::attestation_service$$
secrets:
...
```

Minio service policies: MinIO receives a parameter indicating the volume to store data and the addresses to listen for service and for console panel. One additional parameter is passed, "`--anonymous`", that forces the MinIO log *not to show* administrator and password in log output. When executed with "`Docker run`", for example, it is shown, which poses a major security threat; therefore, it is explicitly hidden.

```
services:
  #####
  # server policy
  - name: minio
    image_name: app_image
    command: minio server {{mnDataDirVolume}}/data --anonymous --address
      "{{mnServiceAddress}}" --console-address "{{mnConsoleAddress}}"
    environment:
      #####
      # application configuration
      # environment variables necessary and with fixed values
      MINIO_UPDATE_MINISIGN_PUBKEY:  "RWTx5Zr1tiHQLwG9keckT0c45M3AGeHD6IvimQHpyRywVW
```

```
        GbP1aVSGav"
      PATH:                         "/usr/bin:/usr/local/sbin:/usr/local/bin:
        /usr/sbin:/sbin:/bin"
      HOME:                         "/root"
      # FSPF environment variable for continued operation
      SCONE_FSPF_MUTABLE:           1
      SCONE_FSS_VERIFICATION_ERROR:  eio
...
      SCONE_NETWORK_SHIELD: {{mnSconeNetshield}}
      SCONE_NETWORK_SHIELD_SERVER_1: {{mnSconeNetshield}}
      SCONE_NETWORK_SHIELD_SERVER_1_PORT: TCP:9000
      SCONE_NETWORK_SHIELD_SERVER_1_CLIENT_AUTH: $$SCONE::ca_certificate$$
      SCONE_NETWORK_SHIELD_SERVER_1_IDENTITY: $$SCONE::server_certificate:privatekey
        :pkcs8:pem$$$$SCONE::server_certificate:crt:pem$$
      SCONE_NETWORK_SHIELD_SERVER_2: {{mnSconeNetshield}}
      SCONE_NETWORK_SHIELD_SERVER_2_IDENTITY: $$SCONE::server_certificate:privatekey
        :pkcs8:pem$$$$SCONE::server_certificate:crt:pem$$
      SCONE_NETWORK_SHIELD_SERVER_2_CLIENT_AUTH: $$SCONE::ca_certificate$$
      SCONE_NETWORK_SHIELD_SERVER_2_PORT: unix:/dev/log
      SCONE_NETWORK_SHIELD_CLIENT_2: {{mnSconeNetshield}}
      SCONE_NETWORK_SHIELD_CLIENT_2_DESTINATION: unix:/run/systemd/journal/dev-log
      SCONE_NETWORK_SHIELD_CLIENT_2_SERVER_AUTH: $$SCONE::ca_certificate$$
      SCONE_NETWORK_SHIELD_CLIENT_2_IDENTITY: $$SCONE::client_certificate:privatekey
        :pkcs8:pem$$$$SCONE::client_certificate:crt:pem$$
      SCONE_NETWORK_SHIELD_CLIENT_2_SERVER_DNS_NAME: confidential
...
      SCONE_NETWORK_SHIELD_SERVER_3: protected
      SCONE_NETWORK_SHIELD_SERVER_3_PORT: unix:/var/run/syslog
      ...
      SCONE_NETWORK_SHIELD_CLIENT_3: protected
      SCONE_NETWORK_SHIELD_CLIENT_3_DESTINATION: unix:/var/run/syslog
...
      SCONE_NETWORK_SHIELD_SERVER_4: protected
      SCONE_NETWORK_SHIELD_SERVER_4_PORT: unix:/var/run/log
      ...
      SCONE_NETWORK_SHIELD_CLIENT_4: protected
      SCONE_NETWORK_SHIELD_CLIENT_4_DESTINATION: unix:/var/run/log
...
    attestation:
      - $$SCONE::attestation_service$$
    pwd: /
images:
  - name: app_image
    volumes:
      - name: set_minio_datadir_volume
        path: {{mnDataDirVolume}}
volumes:
  - name: set_minio_datadir_volume
secrets:
...
```

Keycloak service policies: Keycloak is a large memory-consuming system, hence the memory reserved for the enclave is a full 16GB of RAM, with additional JVM memory parameters to adjust its execution properly. There is a second service "`pypkcsandcas`" that is used for attested execution to generate a PKCS12 credentials set used by Java applications on SSL connections and used in JDBC URL to connect to MariaDB. The first time Keycloak starts, it will set the initial configurations in the database. Once done, the systems administrator can log in to the interface (example `keycloak.neardata.eu:30443/admin/master/console/`) with the initial user and password set via `KC_BOOTSTRAP_ADMIN_USERNAME` and `KC_BOOTSTRAP_ADMIN_PASSWORD`. Once inside, the administrator has to create new user with the **admin** role and remove this one (as explained above).

```
services:
  #####
  # server policy
  - name: keycloak
    image_name: app_image
    command: java -Dkc.home.dir=/keycloak/bin/../ -Djboss.server.config.dir=
    /keycloak/bin/../conf -Djava.util.logging.manager=org.jboss.logmanager
    .LogManager -Dquarkus-log-max-startup-records=10000 -cp /keycloak/bin/../lib/
    quarkus-run.jar io.quarkus.bootstrap.runner.QuarkusEntryPoint --verbose start
    --hostname={{keycloak_fqdn}} {{kcAdditionalParams}}
    environment:
      #####
      # application configuration
      # environment variables necessary and with fixed values
      KEYCLOAK_HOME: /keycloak
      KC_HTTPS_CERTIFICATE_FILE: /tls/keysrv.crt
      KC_HTTPS_CERTIFICATE_KEY_FILE: /tls/keysrv-key.pem
      PROXY_ADDRESS_FORWARDING: "true"
      JAVA_TOOL_OPTIONS: -Xmx{{javaXmx}}
      JAVA_OPTS: '-Xms{{javaXms}} -Xmx{{javaXmx}} -XX:MetaspaceSize={{javaXMetaspace
      }} -XX:MaxMetaspaceSize={{javaXMaxMetaspace}} -Djava.net.preferIPv4Stack=true'
      JAVA_OPTS_APPEND: -Djboss.as.management.blocking.timeout=7200
...
      SCONE_NETWORK_SHIELD: {{kcSconeNetshield}}
      SCONE_NETWORK_SHIELD_SERVER_1: {{kcSconeNetshield}}
      SCONE_NETWORK_SHIELD_SERVER_1_PORT: TCP:{{httpsPort}}
      SCONE_NETWORK_SHIELD_SERVER_1_CLIENT_AUTH: $$SCONE::ca_certificate$$
      SCONE_NETWORK_SHIELD_SERVER_1_IDENTITY: $$SCONE::server_certificate:privatekey
        :pkcs8:pem$$$$SCONE::server_certificate:crt:pem$$
...
    attestation:
      - $$SCONE::attestation_service$$
    pwd: /keycloak
...
  #####
  # auxiliary program to register policy with PKCS12 credentials for Keycloak to
  # access MariaDB
  - name: pypkcsandcas
    image_name: transformer_image
    command: python3 /python/pkcsandcas.py
    pwd: /python
    environment:
      #####
      # program configuration
      MARIADB_CLIENT_CERT: "$$SCONE::MARIADB_CLIENT_CERT.crt$$"
      CAS_URL: {{cas.cas.cas_url}}
      CAS_SESSION: {{session}}
      SFX_PKCS_SESSION: {{{suffixPkcsSession}}}
      PKCSPWD: $$SCONE::MariadbPkcs12Password$$
      PKCSALIAS: MARIADB_CLIENT_CERT
      PKCSCACRT: /tls/i_mariadb-ca.crt
      PKCSCERT: /tls/i_mariadb-client.crt
      PKCSCERTPRIV: /tls/i_mariadb-client.key
      CASCLICERT: /tls/cert.pem
      CASCLIKEY: /tls/key.pem
      POLTMPL: /tls/pol.template.yaml
      SECRETSTORE: MariadbPkcs12
    attestation:
      - $$SCONE::attestation_service_saver$$
...
  - name: kc-client-userpass
    command: python3 kc-client-injection.py
    pwd: /python
    environment:
...
  - name: kc-loginpage-lt-multiproc
    command: python3 kc-multiprocessinglithops.py
    pwd: /python
    environment:
...
###
# services images: configurations, files and secret injections for each service
# - app_image: Keycloak configurations
# - transformer_image: PKCS generator configurations
images:
  - name: app_image
    injection_files:
        - path: /tls/scn-keycloak-ca.crt
          content: $$SCONE::Keycloak_CA_Cert.chain$$ # Export this session's CA
                                                     # certificate & chain
        - path: /tls/keysrv.crt
          content: $$SCONE::Keycloak_Server_Cert.crt$$
        - path: /tls/keysrv-key.pem
          content: $$SCONE::Keycloak_Server_Cert.key$$
        - path: /tls/scn-keycloak-client.crt
          content: $$SCONE::Keycloak_Client_Cert.crt$$
        - path: /tls/scn-keycloak-client.key
          content: $$SCONE::Keycloak_Client_Cert.key$$
        ###
        # PKCS12 file with client MariaDB credentials used by Java applications
        # currently generated by pkcsandcas.py auxiliary program
        - path: /tls/mariadb.pkcs12
          content: $$SCONE::Mariadb_Pkcs12:bin$$
    volumes:
      - name: prodtmp
        path: /production/hashes/tmp
  - name: transformer_image
    injection_files:
        ###
        # mariadb credentials imported
        - path: /tls/i_mariadb-ca.crt
          content: $$SCONE::MARIADB_CA_CERT.chain$$
        - path: /tls/i_mariadb-client.crt
          content: $$SCONE::MARIADB_CLIENT_CERT.crt$$
        - path: /tls/i_mariadb-client.key
          content: $$SCONE::MARIADB_CLIENT_CERT.key$$
        - path: /tls/pol.template.yaml
          content: |
            name: POLNAME
            version: "0.3.10"
            access_policy:
              read:
                - CREATOR
              update:
                - CREATOR
            security:
              attestation:
                tolerate: [debug-mode, hyperthreading, outdated-tcb,
                           insecure-configuration, software-hardening-needed]
                ignore_advisories: "*"
            secrets:
              - name: SECRETSTORE
                kind: binary
                value: KEYBIN
                export:
                - session: EXPSESS
    volumes:
      - name: prodtmp
        path: /production/hashes/tmp
secrets:
...
  - name: attestation_service_saver
    import:
      session: {{image_policy_name}}
      secret: mrsigner_mrenclave_versions
      cas_url: {{cas.image.cas_url}}
      cas_key: {{cas.image.cas_key}}
  - name: MARIADB_CA_CERT
    import:
      session: {{namespace}}/secrets
      secret: MARIADB_CA_CERT
  - name: MARIADB_CLIENT_CERT
    import:
      session: {{namespace}}/secrets
      secret: MARIADB_CLIENT_CERT
  - name: MARIADB_CLIENT_KEY
    import:
      session: {{namespace}}/secrets
      secret: MARIADB_CLIENT_KEY
  ###
  # obtained from the attested program pkcsandcas.py, which gets client
  # credentials from MariaDB, makes a PCKCS12 and uploads onto the CAS
  # under the session: {{namespace}}-{{{suffixPkcsSession}}}
  - name: Mariadb_Pkcs12
    import:
      session: {{namespace}}-{{{suffixPkcsSession}}}
      secret: MariadbPkcs12
volumes:
  - name: prodtmp
```

MariaDB service policies: The database supporting Keycloak is also prepared in confidential computing. The sharing with Keycloak policies of the database client credentials, that is automatically generated by CAS, allows for the subsequent generation of PKCS12 credentials done by the auxiliary attested application as initialization container in Keycloak cluster setup. MariaDB's administrative `root`'s user password is also generated automatically by CAS. The complete installation takes 3 phases: database bootstrap, without any external access; initial users and schemes setup, to create users and databases; and startup with network support. This process is done only once; the next time the system is started, it will directly load the database server.

```
services:
  #####
  # server policy
  - name: db
    image_name: db_image
    command: mysqld --innodb-use-native-aio=0 --innodb-flush-method=fsync
    attestation: $$SCONE::attestation_service_usr_bin_mysqld$$
    pwd: /
    environment:
      MYSQL_ROOT_PASSWORD: "$$SCONE::MYSQL_ROOT_PASSWORD$$"
      MYSQL_ALLOW_EMPTY_PASSWORD: ""  # Empty means 'false', anything else 'true'
      MYSQL_RANDOM_ROOT_PASSWORD: ""  # Empty means 'false', anything else 'true'
      SCONE_NETWORK_SHIELD: unprotected
      SCONE_NETWORK_SHIELD_SERVER_1: protected
...
  - name: bootstrap
    image_name: bootstrap_image
    command: mysqld --bootstrap --basedir=/usr --datadir=/var/lib/mysql
      --log-warnings=0 --plugin-dir=/usr/lib/mariadb/plugin --innodb-use-native-aio=0
      --user=mysql --max_allowed_packet=8M --net_buffer_length=16K
      --default-storage-engine=innodb
    attestation: $$SCONE::attestation_service_usr_bin_mysqld$$
    pwd: /
    environment:
      MYSQL_ROOT_PASSWORD: "$$SCONE::MYSQL_ROOT_PASSWORD$$"
      MYSQL_ALLOW_EMPTY_PASSWORD: ""  # Empty means 'false', anything else 'true'
      MYSQL_RANDOM_ROOT_PASSWORD: ""  # Empty means 'false', anything else 'true'
      SCONE_NETWORK_SHIELD: protected
  - name: db_before_setup
    image_name: db_image
    command: mysqld --innodb-use-native-aio=0 --innodb-flush-method=fsync
      --skip-networking
    attestation: $$SCONE::attestation_service_usr_bin_mysqld$$
    pwd: /
    environment:
      SCONE_LOG: debug
      SCONE_NETWORK_SHIELD: protected
      SCONE_NETWORK_SHIELD_UNSUPPORTED_PROTOCOLS: refuse
      SCONE_NETWORK_SHIELD_SERVER_1: protected
...
  - name: create_user
    image_name: mysql_client_setup_image
    command: ["mysql", "-e", "source /etc/create-user.sql;"]
    attestation: $$SCONE::attestation_service_usr_bin_mysql$$
    pwd: /
    environment:
      SCONE_LOG: debug
      # Only explicitly configured connections are permitted, all others are refused
      SCONE_NETWORK_SHIELD: protected
      SCONE_NETWORK_SHIELD_UNSUPPORTED_PROTOCOLS: refuse
      SCONE_NETWORK_SHIELD_CLIENT_1: protected
...
images:
  - name: mysql_client_setup_image
    injection_files:
      - path: /etc/create-user.sql
        content: {{{sqlInitScript}}}
  - name: bootstrap_image
    volumes:
      - name: encrypted_datadir_volume
        path: /var/lib/mysql
        update_policy: no_rollback_protection
      - name: var_tmp
        path: /var/tmp
        update_policy: no_rollback_protection
      - name: external_datadir_volume
        path: /external
        update_policy: no_rollback_protection
  - name: db_image
    volumes:
      - name: encrypted_datadir_volume
        path: /var/lib/mysql
        update_policy: no_rollback_protection
      - name: var_tmp
        path: /var/tmp
        update_policy: no_rollback_protection
      - name: external_datadir_volume
        path: /external
        update_policy: no_rollback_protection
    injection_files:
      - path: /etc/my.cnf
        content: {{{etcMyCnf}}}
      - path: /etc/keys.txt
        content: "1;$$SCONE::mysql_encryption_key16.hex$$;$$SCONE::mysql_encryption
          _key32.hex$$"
      - path: /etc/mariadb-ca.crt
        content: $$SCONE::MARIADB_CA_CERT.chain$$
      - path: /etc/server.crt
        content: $$SCONE::mariadb.crt$$
      - path: /etc/server.key
        content: $$SCONE::mariadb.key$$
      - path: /etc/client.crt
        content: $$SCONE::MARIADB_CLIENT_CERT.crt$$
      - path: /etc/client.key
        content: $$SCONE::MARIADB_CLIENT_CERT.key$$
secrets:
...
  - name: mysql_encryption_key16
    kind: binary
    size: 16
  - name: mysql_encryption_key32
    kind: binary
    size: 32
  - name: MYSQL_ROOT_PASSWORD
    import:
      session: {{session_secrets}}
      secret: MYSQL_ROOT_PASSWORD
  - name: mariadb-key
    import:
      session: {{session_secrets}}
      secret: mariadb-key
  - name: mariadb
    import:
      session: {{session_secrets}}
      secret: mariadb
volumes:
  - name: encrypted_datadir_volume
  - name: var_tmp
  - name: external_datadir_volume
```

SCONECTL is a great choice to manage the confidential mesh of services lifecycle; it will keep the policies up-to-date and the secrets fresh. All configurations have been made available at `https://github.com/`

`neardata-eu/scone-artifacts/`.

5 Final Release of the Data Broker

This section describes the components of the Data Broker. To describe these components, we will utilize the Software Design Specification (SDS) standard (IEEE Standard 1016), which will help us explain their purpose and function, as well as their position within the high-level architecture. This section focuses exclusively on components that control data access and provide security measures within NEARDATA.

Since we have made slight changes to the template used to describe our tools compared to previous projects, we will first summarize the adopted methodology. Next, we will describe each component, its main functionalities, and its high-level architecture. Finally, we will provide a detailed description of the activities performed and the enhancements introduced in the final release of the tools over the last twelve months of the project. A specific subsection is dedicated to analysing the performance evaluation. The second release of the tools.

Identification	The unique name for the component and its location in the system
Type	A module, a subprogram, a data file, a control procedure, a class, etc.
Purpose	Function and performance requirements implemented by the design component, including derived requirements. Derived requirements are not explicitly stated in the SRS, but are implied or adjunct to formally stated SDS requirements.
Function	What the component does, the transformation process, the specific inputs that are processed, the algorithms that are used, the outputs that are produced, where the data items are stored, and which data items are modified.
High level Architecture	The internal structure of the component, its constituents, and the functional requirements satisfied by each part.
Dependencies	How the component's function and performance relate to other components. How this component is used by other components. The other components that use this component. Interaction details such as timing, interaction conditions (such as order of execution and data sharing), and responsibility for creation, duplication, use, storage,and elimination of components.
Interfaces	Detailed descriptions of all external and internal interfaces as well as of any mechanisms for communicating through messages, parameters, or common data areas. All error messages and error codes should be identified. All screen formats, interactive messages, and other user interface components (originally defined in the SRS) should be given here.
Data	For the data internal to the component, describe the representation method, initial values, use, semantics, and format. This information will probably be recorded in the data dictionary.
Summary of the First Release	A summary of the new features introduced during Y1 within the NEARDATA project.
Summary of the final Release	A description of the implemented improvements in the service to achieve the first release of the runtime environment.
Release Version & Repository	The software version released and the repository from where it can be downloaded.

Table 8: Final Release of the Data Broker

5.1 Data Broker and Security Mechanisms

The security mechanisms and tools presented here offer a range of protective measures to ensure confidentiality, integrity, and freshness when applied within the context of NEARDATA.

5.1.1 Runtime Security

In this section, we will present the different security measures we implemented in order to run applications in a secure fashion. In order to achieve this, we follow the confidential computing paradigm which focuses on the protection of applications. Hence, we require that the mechanisms we implement protect the application's:

1. **Confidentiality**, i.e., no other entity, like a root user, can read the data, the code, or the secrets of the application in memory, on disk or on the network,
2. **Integrity**, i.e., no other entity, like the hypervisor, can modify the data, or at least any modifications are detected in the memory, on disk or on the network, and
3. **Consistency**, i.e., the application always reads the value that was written last - both in memory as well as on disk and on the network.

5.1.2 Hardware Support

Currently, there are multiple classes of hardware support for confidential computing available, representing a significant evolution from the original two-class model. The current mechanisms can be classified as follows.

- **VM-level protection: Encrypt entire Virtual Machines (VMs) using technologies like AMD SEV-SNP and Intel TDX**
- **Process-level enclaves: Encrypt individual services or processes inside enclaves (Intel SGX)**
- **Architecture-level realms: ARM's Confidential Compute Architecture (CCA) introducing dynamic realms**

The Latest Developments in Hardware Technologies (2024-2025)

1. **Intel Technologies - 2024-2025 Developments**
   - **TDX Production Rollout:** Intel TDX hardware shipments reached production scale in 2024, with major cloud service providers including Microsoft Azure, Google Cloud, and AWS beginning widespread deployment [6]. This marks the transition from experimental to enterprise-ready confidential computing infrastructure.
   - **Enterprise Adoption:** Unlike the limited adoption seen with SGX, TDX is experiencing rapid uptake due to its VM-level protection model that requires no application modifications. The technology leverages the proven secure arbitration mode (SEAM) architecture while utilizing SGX quoting enclaves for attestation, creating a hybrid approach that maximizes compatibility [7].
2. **AMD Advancements - 2024-2025 Security Updates**
   - **Vulnerability Research and Mitigations:** The BadRAM attack demonstrated in 2024 by Chen et al. exposed fundamental challenges in memory protection, showing how $10 hardware modifications could compromise SEV-SNP protections. This prompted AMD to release enhanced firmware updates and additional security guidelines throughout 2024-2025 [8].
   - **SEV-SNP Production Hardening:** AMD has focused on strengthening production deployments with improved attestation mechanisms and enhanced memory integrity checks. The company has also worked closely with cloud providers to implement additional detection mechanisms for hardware-based attacks.
3. **ARM CCA - 2024-2025**
   - **First Commercial Implementations:** 2024 marked the first commercial availability of ARM CCA-enabled processors, with initial deployments in edge computing and mobile applications. The Realm-based architecture has shown particular promise in IoT and automotive applications where dynamic security requirements are critical.
   - **Ecosystem Development:** ARM has accelerated ecosystem development in 2024-2025, with major cloud providers announcing support for CCA-based services and multiple OS vendors integrating Realm support into their platforms. This rapid adoption is driven by the architecture's flexibility and standardized approach compared to vendor-specific solutions.

5.1.3 Encrypted VMs vs. Enclaves

When running applications in untrusted environments such as public clouds, there are usually multiple stakeholders involved. This includes:

- a host admin that maintains the host Operating System (OS)
- a container/service admin that takes care of the services and the containers.

1. **Encrypted VMs Approach** In the context of an AI application in NEARDATA, an encrypted VM would be used to represent, e.g., a worker node performing training or inference. The trusted computing base would not only include the AI application itself but we would also need to trust:

   - the operating system and the OS admin and
   - the container/service admin.

   One approach is to execute each container/process in a separate encrypted VM. This would reduce the size of the trusted computing base (TCB) since the host admin would not be part of the trusted computing base anymore. However, the container/service admin and the operating system within the VM and its OS admin would still be part of the TCB.

2. **Enclaves Approach** In contrast to VMs, enclaves permit us to reduce the size of the trusted computing base to the application/process itself: we can remove all admins and all code outside of the application from the trusted computing base. Note that our approach will help to protect the files of an AI application and also the network if needed, i.e., a service admin will only see encrypted files and will not know any application secrets.

3. **Realms Approach (ARM CCA)** RME introduces a new kind of confidential compute environment called a realm. Any code or data belonging to a realm, whether in memory or in registers, cannot be accessed or modified by code or agents outside the TCB:

5.1.4 Isolation and Cooperation

The advantage of the enclave-based approach is that one can protect services from each other. A service has only access to its own enclave and to its files but not to the data/files of other enclaves.

Using encrypted VMs, one would need to run each service in a separate VM with its own operating system which increases the TCB as explained above. This would also increase the resource usage since each container would come with its own operating system which is also impractical in the context of edge devices and their limiting processing capacity.

SCONE added a new network shield to support confidential service meshes. With the help of our CAS as part of the Data Broker, which we will explain more in detail later in this document, services of an application can cooperate and implicitly attest each other via TLS: a service can only establish a TLS connection with another service of the application if that service executes inside of an enclave, its code was not modified and the filesystem is in the correct state. This is an important property as it ensures that only trusted collaborator nodes in federated learning can establish connections to the aggregator node and vice versa.

5.2 SCONE Overview

In order to use Intel SGX, programmers need to download and install the Intel SGX SDK and extend their application to use the new instructions. In its original design, the framework creators only envisioned the use of Intel SGX for secret generation, i.e., that only a single function runs within an enclave without any further communication to the untrusted outside world. However, running an entire process in an enclave imposes the following challenges:

1. An application running in an enclave cannot perform any system calls such as writing/reading to/from files or a network socket, etc. If a system call needs to be executed, the enclave execution must be first paused, the system call is then executed and the application resumed afterwards inside the enclave. This imposes a huge overhead as applications performing hundreds of system calls per second are constantly.

2. Applications must be instrumented in order to run in an enclave which comprises allocation of enclave memory, loading program as well as data code into the enclave memory and launching. This is a cumbersome task as it requires modifications of all existing applications.

To avoid these cumbersome tasks, we will base our work on SCONE, a framework that transforms applications in confidential applications without any developer's effort.

In a nutshell, SCONE consists of a cross-compiler which adds all the necessary instrumentation as well as starter code etc. to let the process run in an Intel SGX enclave. The following tables list the advantages of SCONE compared to the use of the Intel SGX SDK.

- **Multi-Platform Binary Support:** SCONE has evolved significantly beyond its original Intel SGX focus with enhanced compatibility. SCONE version 5.7+ now supports both musl as well as glibc-based binaries, dramatically expanding compatibility across Linux distributions including Ubuntu, CentOS, RHEL (glibc) and Alpine Linux (musl). The platform also mentions future extension to support vendors other than Intel CPUs such as AMD, ARM etc.
- **Enterprise Production Scaling:** 2024 saw unprecedented enterprise adoption with SCONE being selected as the key framework to provide confidential computation to major projects including Flower ML, Pravega, and Lithops, demonstrating its maturity for large-scale deployments.
- **Zero Trust Architecture Integration:** In November 2024, SCONE introduced comprehensive Zero Trust Architecture support, aligning with modern cybersecurity frameworks and enabling seamless integration with enterprise security policies.
- **Confidential Landing Zones:** A major 2024 collaboration between Accenture, Intel, and Scontain resulted in the design of confidential landing zones, providing enterprise-grade deployment patterns for confidential computing in cloud environments.

In short, SCONE is a cross-compiler that adds the necessary instrumentation, starter code, etc. to enable processes to run in an Intel SGX enclave. It now also supports multiple TEE architectures.

5.2.1 SCONE Runtime

The objective of SCONE is to build and run applications in a confidential environment with the help of Intel SGX (Software Guard eXtensions). In a nutshell, our objective is to run applications such that data is always encrypted, i.e., all data at rest, all data on the wire as well as all data in main memory is encrypted. Even the program code can be encrypted such as the Python code files typically used for AI-based applications. SCONE helps to protect data, computations and code against attackers with root access.

The aim of SCONE is to make it as easy as possible to secure existing applications such as TensorFlow, Pytorch, etc. typically used in the machine learning domain. Hence, switching to SCONE is simple as applications do not need to be modified. SCONE supports the most popular programming languages like JavaScript, Python - including PyPy, Java, Rust, Go, C, and C++ but also ancient languages such as Fortran. Avoiding source code changes helps to ensure that applications can later run on different trusted execution environments. Moreover, there is no risk for hardware lock-in nor software lock-in - even into SCONE itself.

SCONE version 5.7.0 represented a major milestone with stability improvements and a large set of new features that laid the foundation for modern confidential computing applications:

1. Multi-Platform Binary Support:
   - Glibc and Musl Support: SCONE and sconify_image now support both musl as well as glibc-based binaries, dramatically expanding compatibility across Linux distributions including Ubuntu, CentOS, RHEL (glibc) and Alpine Linux (musl)
   - Dynamic Library Loading: Enhanced support for runtime extensions and dynamic library loading with virtual syscalls for handling dlopen in glibc
   - ELF File System: Introduction of elf fs for optimized handling of executable files
2. Network Shield Introduction:
   - Confidential Service Meshes: Added a new network shield to support confidential service meshes, enabling transparent TLS encryption for applications that don't support TLS natively
   - Runtime Extensions: Support for runtime extensions with both glibc and musl platforms
3. Enhanced CAS Features:
   - Audit Logging: Added enclave identities to audit log for comprehensive security monitoring
   - Connection Reliability: Implemented reconnect to CAS functionality, keep-alive messages, and configurable connection timeouts

- Performance Optimization: Improved dynamic library loader and support for position-dependent executables

SCONE 5.8.0 introduced comprehensive enterprise features focused on governance, security, and operational excellence.

1. Governance and Access Control:
   - Governance Access Policy Rules: Comprehensive access control mechanisms for enterprise deployments
   - Session Encryption and Signing: Support for session encryption keypair, multiple session signers, and session signature verification
   - Role-Based Access: Allow pre-configured session creators and signer-based access controls
2. Air-Gapped Operations:
   - Platform-Based Attestation: Support for air-gapped operations where applications can run without access to external attestation services
   - Offline CAS Attestation: Capability to perform attestation in disconnected environments
   - Database Integrity: CAS database integrity checks on startup and improved snapshotting algorithms
3. Azure Integration and Cloud Services:
   - Azure Key Vault (AKV) Integration: Complete AKV secret backend support including setting and deleting AKV secrets
   - Microsoft Attestation (MAA) Tokens: Enhanced integration with Azure attestation services
   - Azure PCCS Cache Support: Support for Azure Platform Configuration and Control Service cache
4. Network Shield Enhancements:
   - Unix Sockets Support: Network Shield now supports Unix sockets in addition to TCP connections
   - Advanced Logging: Improved Network Shield logging and error codes for better troubleshooting
   - Socket Security: Tolerate Insecure Unix Credentials Flag for Enhanced Socket Security
5. DCAP and Attestation Improvements:
   - DCAP API v4: Support for latest DCAP data models and API versions
   - PCK Certificate Management: Automatic PCK certificate renewal and provisioning capabilities
   - TCB Information: Enhanced platform TCB information exposure and state management
6. Development and deployment:
   - Binary File System (binary-fs): Enhanced binary-fs support for various applications including MariaDB, nginx, memcached
   - Container Images: Addition of numerous curated images including Apache Flink, TensorFlow Lite, PyTorch, and PHP variants
   - Rust Ecosystem: Comprehensive Rust support with multiple version upgrades from 1.57.0 to 1.70.0

SCONE 5.9.0 introduces advanced security features, production optimizations, and enhanced enterprise capabilities:

1. Advanced Security Features:
   - SGX Local Attestation: Added SGX Local Attestation of service enclaves with must_be_sgx_local attestation capability
   - Secure Rollback Protection: Implemented secure rollback-protected CAS provisioning for enhanced security

- REST API Signing: Complete REST API signing with key selection parameters for secure communications
- Memory Protection: Activated mprotect by default for enhanced memory protection

2. Enterprise Namespace Management:
   - Root Namespace: Added default root namespace and namespace_hash for better organization
   - Namespace Paths: Support for root namespace paths enabling hierarchical secret management
   - Audit Trail: Enhanced audit logging with get-audit-log-checkpoints command for compliance tracking
3. File System and Storage Enhancements:
   - Authenticated-Only Volumes: New authenticated-only volume files for enhanced data protection
   - FSPF Updates: Added fspf_update_policy to main FSPFs (File System Protection Files)
   - WAL Support: Write-Ahead Logging (WAL) for improved data consistency and recovery
   - Migration to Rust: File system service (fss) migration to Rust for improved performance and security
4. Production and Performance Optimizations:
   - Intel SGX SDK Update: Upgraded from Intel SGX SDK 2.20 to 2.23 for latest security and performance improvements
   - Runtime Metrics: Introduction of comprehensive runtime metrics for monitoring and optimization
   - Performance Fixes: Resolved FSPF v1/v2 loading performance regression and optimized system calls
   - Memory Management: Enhanced memory handling with MADV_FREE support and optimized clock_gettime
5. Container and Cloud Enhancements:
   - Health Checks: Added container healthcheck for CAS, LAS, and PCCS components
   - Debian 12 Support: LAS images now support Debian 12 as base image
   - Non-Root Users: Enabled non-root user support for scone.cloud (LAS and CAS)
   - Docker Integration: Enhanced Docker host checking and configuration validation
6. Development and API Improvements:
   - Secret Management: New scone_get_secret_version() API for enhanced secret handling
   - Public-Key Secrets: Added public-key secrets to session language 0.3.11
   - External Signer Support: Enhanced scone-signer with external signer key support and builtin-signer argument
   - Rust Ecosystem: Major Rust upgrade from 1.70.0 to 1.75.0 with improved SGX report verification
7. Azure and Cloud Integration:
   - Azure PCCS Configuration: User configuration of Azure PCCS usage for flexible cloud deployment
   - Provisioning Enhancements: Added provisioning_cas_owner_certificate_chain to CAS attestation report v4
8. Debugging and Development:
   - Enhanced Logging: Allow debug and trace logs in production builds for better troubleshooting
   - Context Switch Mode: Introduction of context switch mode for improved enclave management

- System Call Support: Enhanced system call support including epoll_pwait2 and readlinkat for injected files

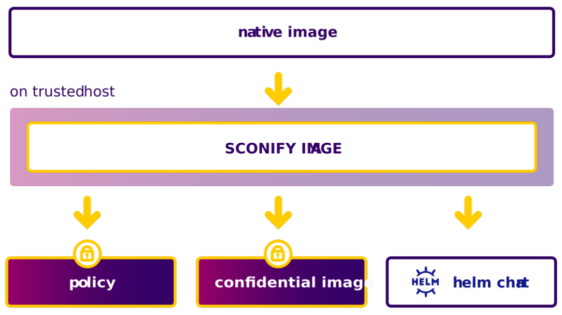
These three versions represent SCONE's evolution from a research-oriented SGX framework to a production-ready, enterprise-grade confidential computing platform capable of supporting complex multi-stakeholder workflows, air-gapped operations, and comprehensive cloud integration while maintaining the highest security standards.

In a nutshell, SCONE consists of a cross-compiler which adds all the necessary instrumentation as well as starter code etc. to let the process run in an Intel SGX enclave, while now extending support to multiple TEE architectures. SCONE Platform Modernization (2024-2025):

- **Next-Generation CAS Features:** Enhanced Configuration and Attestation Service with advanced audit logging and compliance tracking
- **Cloud-Native Integration:** Streamlined Kubernetes deployment with confidential containers support
- **Multi-Stakeholder Workflow Engine:** Advanced support for complex federated learning and multi-party computation scenarios
- **Performance Optimization:** Significant improvements in enclave initialization and runtime performance

In the following, we will present the different SCONE subcomponents, starting with the runtime which will be added to the binaries through the cross compiler. We then also present the Sconify tool which automates many steps such as the cross compilation in order to convert a native application into a confidential one. The section concludes with the CAS, the Configuration and Attestation service which is a service that is used in order to attest services running in enclaves as well as to perform secret provisioning.

Table 9: SCONE Sconify Tool - Evolution and Current Capabilities

Identification	**SCONE Sconify Tool**
Type	A CLI tool that turns a native Docker image into a confidential image that uses the SCONE runtime
Purpose	The SCONE sconify tool transforms standard Docker images into confidential containers by integrating the SCONE runtime. It enables existing applications to execute securely within hardware-protected enclaves without requiring code modifications.
Function	The sconify tool performs comprehensive binary transformation and system integration: **Binary Processing:** • Re-links applications with SCONE libc (both glibc and musl variants) • Automatically validates PIE compilation (position independent code) • Supports native musl and glibc-based images with automatic detection • Enhanced language support: C, C++, Python, Java, Go, Node.js, Rust, and JavaScript • Improved Go support with both gcc-go and standard Go compiler detection • Intelligent interpreter replacement for runtime environments **Advanced System Integration:** • Automatic dependency resolution with enhanced library detection • Comprehensive filesystem encryption with region-based protection policies • Binary filesystem (binary-fs) support for optimized executable handling • Network Shield integration for confidential service mesh capabilities **Modern DevOps Integration:** • Enhanced Dockerfile generation with multi-platform support • Automated SCONE policy generation and CAS submission • Advanced Helm chart generation for Kubernetes deployment • Support for air-gapped operations and governance policies
High level Architecture	The sconify tool operates as an advanced CLI tool with enterprise integration  native image on trustedhost SCONIFY IMAGE policy confidential imag HELM helm chat
Dependencies	The sconify tool is distributed as a containerized solution with comprehensive dependency management: • Complete Docker image packaging with all required libraries • Support for both Intel SGX SDK 2.23 and legacy versions • Integrated Rust toolchain (1.75.0) for enhanced performance • Multi-architecture support for x86_64 and future ARM compatibility
Interfaces	Multi-modal interface support: • Enhanced CLI with comprehensive argument file support and escaping • REST API integration for automated DevOps pipelines • Kubernetes operator integration for cloud-native deployments • Direct CAS integration for policy and secret management
Data	The SCONE runtime uses SCONE policies as well as Environment variables to configure the application correctly.

Continued on next page

Table 9 Continued from previous page

Identification	**SCONE Sconify Tool**
Summary of the First Release	The first release of the SCONE runtime regarding NEARDATA includes the following improvements: created first draft/implementation of the network shielding layer, improved performance with regards to system call behaviour, integration with the latest version of CAS.
Summary of the Final Release	SCONE sconify tool evolution (5.7.0 - 5.9.0) represents a comprehensive transformation into an enterprise-grade confidential computing platform: **Version 5.7.0 Breakthroughs:** • Universal glibc/musl binary support enabling cross-platform deployment • Network Shield integration for confidential service mesh architectures • Enhanced dynamic library loading and runtime extension capabilities **Version 5.8.0 Enterprise Features:** • Comprehensive governance and access control mechanisms • Air-gapped operation support with platform-based attestation • Azure cloud integration (AKV, MAA) and enterprise DevOps workflows • Advanced container image support and curated application templates **Version 5.9.0 Production Excellence:** • Advanced security with SGX Local Attestation and rollback protection • Enterprise namespace management and hierarchical organization • Rust migration for enhanced performance and memory safety • Production-ready health checks and operational capabilities The tool now supports comprehensive enterprise deployments with advanced security, governance, and operational features essential for mission-critical confidential computing applications.
Release Version & Repository	**Current Version:** SCONE 5.9.0 (Latest Stable) **Previous Versions:** 5.8.0, 5.7.0 (Major feature releases) **Repository Access:** • Official Documentation: https://sconedocs.github.io/ • Docker Registry: registry.scontain.com/ • Azure Marketplace: SCONE Confidential Computing Platform • Public CAS Instances: 5-9-0.scone-cas.cf (Production) **Release Channels:** • Stable: Production-ready releases with enterprise support • Edge: edge.scone-cas.cf (Latest nightly builds for testing) **Enterprise Support:** • Commercial licenses available through Scontain UG • Community version available for development and evaluation • Integration support for Azure, AWS, and Google Cloud platforms

Detailed Description of Activities for the Final Release

The latest release of the SCONE Sconify Tool (version 5.9.0) represents a significant evolution, transforming the platform from version 5.6.0 into an enterprise-grade confidential computing solution. The development journey began with the establishment of critical foundational capabilities, including support for DCAP quote generation, compatibility with simulated mode on AMD chips and enhanced runtime stability with improved logging mechanisms. This baseline version introduced vital security features, including attestation vulnerability options, complete image encryption capabilities and Helm chart parameter overriding. It also supported progress bar indicators and comprehensive standardised host testing frameworks, forming the technical basis for subsequent major enhancements.

Version 5.7.0 marked the first major breakthrough by implementing universal binary compatibility supporting both glibc and musl libraries, eliminating the platform lock-in that had previously limited enterprise adoption across diverse Linux distributions. This release introduced the revolutionary Network Shield technology, enabling legacy applications without native TLS support to participate seamlessly in confidential service meshes, while enhancing dynamic library loading with virtual syscalls for dlopen functionality in glibc environments. The development team also implemented ELF filesystem support for optimized executable handling, automatic CAS reconnection with keep-alive mechanisms, and comprehensive runtime extensions that significantly improved reliability and performance for production workloads.

The enterprise integration phase, version 5.8.0, focused on addressing complex organizational requirements by architecting a comprehensive governance framework featuring role-based access controls, session encryption and signing capabilities, and extensive audit logging mechanisms essential for regulated industries. This release implemented groundbreaking air-gapped operations support with platform-based attestation for high-security environments, developed extensive Azure cloud integration including Key Vault backend and Microsoft Attestation service compatibility, and enhanced container orchestration with advanced Helm chart generation and Kubernetes operator integration. The development team significantly expanded language support to include the comprehensive Rust ecosystem with multiple version upgrades, improved Go compiler detection beyond gcc-go limitations, and introduced sophisticated container image templates including Apache Flink, TensorFlow Lite, and various curated application frameworks.

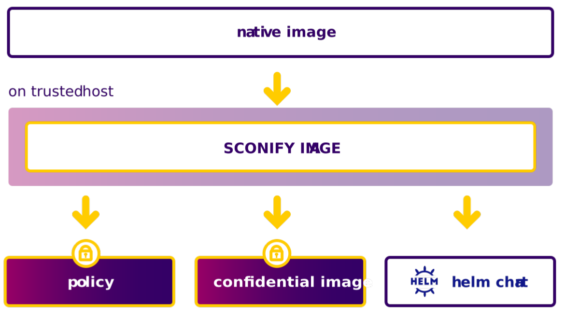
The latest phase, version 5.9.0, achieved production excellence by implementing advanced security architectures including SGX Local Attestation and secure rollback-protected provisioning, developing hierarchical namespace management with root namespace support enabling enterprise-scale organization and access control, and migrating core file system services to Rust for enhanced memory safety and performance optimization. This release integrated Intel SGX SDK 2.23 with production-ready health checks and comprehensive operational monitoring, enhanced binary filesystem support with authenticated-only volume capabilities and Write-Ahead Logging, and introduced runtime metrics for comprehensive performance analysis. The cumulative evolution from version 5.6.0 through 5.9.0 transformed the SCONE Sconify Tool from a research-oriented SGX transformation utility into a comprehensive enterprise-grade confidential computing platform capable of supporting mission-critical deployments with sophisticated governance mechanisms, multi-cloud integration, advanced security features, and operational excellence while maintaining full application compatibility and performance characteristics essential for large-scale enterprise adoption.

Identification	**SCONE Cross Compiler**
Identification	SCONE Cross Compiler
Type	A comprehensive CLI tool that compiles source code into confidential applications with advanced multi-platform and multi-architecture support
Purpose	The SCONE cross compiler enables application developers to compile source code for confidential execution within TEEs, automatically integrating necessary instructions for Intel SGX, with future support for Intel TDX and AMD SEV-SNP architectures.
Function	The cross compiler performs comprehensive source code compilation with enhanced capabilities: **Language Support:** • C/C++ with enhanced optimization for performance • Fortran for scientific computing applications • Go with both gccgo and standard compiler support • Rust with integrated toolchain (v1.75.0) for memory-safe applications • Python through interpreter compilation and filesystem protection • JavaScript/Node.js for web-based applications **Advanced Features:** • Automatic starter code injection for enclave initialization • Enhanced memory management optimization • Multi-threading support for parallel processing • Cross-compilation for heterogeneous architectures • Integration with SCONE's Network Shield for secure communication **Compilation Optimizations:** • Position Independent Executable (PIE) code generation • Enhanced dynamic library linking capabilities • Optimized binary generation for enclave environments • Support for both static and dynamic linking strategies
High level Architecture	The cross compiler operates as an advanced CLI tool with enterprise integration capabilities, supporting modern development workflows and multi-platform deployment requirements across diverse computing infrastructure.

Identification	**SCONE Cross Compiler**
Dependencies	Enhanced dependency management for complex development requirements: • Docker-based packaging with comprehensive library support • Intel SGX SDK 2.23 integration for latest hardware capabilities • Universal musl and glibc support for diverse Linux distributions • Rust toolchain integration for memory-safe applications • Cross-platform compilation tools for ARM and x86_64 architectures • Standard development libraries and framework integration support
Interfaces	Multi-modal interface support for modern development ecosystems: • Enhanced CLI with comprehensive build system integration • IDE plugin compatibility for popular development environments • CI/CD pipeline integration for automated confidential application builds • Integration with SCONE sconify tool for end-to-end deployment workflows
Data	Advanced data handling for secure compilation requirements: • Source code protection during compilation process • Secure artifact generation with integrity verification • Build cache management for large-scale project compilation • Encrypted intermediate file handling for sensitive codebases
Summary of the First Release	Initial SCONE cross compiler (pre-5.7.0) provided fundamental compilation capabilities with basic SGX support. Key features included: • Limited language support (primarily C/C++ and basic Python) • Basic musl library integration with manual dependency management • Simple Go support restricted to gccgo compilation • Basic enclave memory allocation without optimization • Manual build processes without automation capabilities
Summary of the Final Release	SCONE cross compiler evolution (5.7.0 - 5.9.0) represents comprehensive advancement in confidential computing compilation capabilities: **Version 5.7.0 - Foundation Enhancement:** • Universal glibc/musl support enabling cross-platform deployment • Enhanced Go compiler support beyond gccgo limitations • Dynamic library optimization and virtual syscall integration • Improved runtime extension capabilities **Version 5.8.0 - Enterprise Integration:** • Advanced build system integration supporting large-scale projects • Comprehensive Rust toolchain with multiple version upgrades (1.57.0 to 1.70.0) • Enhanced optimization flags for performance-critical applications • Integration with governance frameworks for enterprise development **Version 5.9.0 - Production Excellence:** • Intel SGX SDK 2.23 integration with latest hardware optimizations • Advanced memory management for large application compilation • Enhanced cross-compilation capabilities for diverse architectures • Production-ready build artifacts with comprehensive integrity verification • Rust toolchain upgrade to version 1.75.0 with enhanced capabilities The compiler now supports complete enterprise development lifecycle from secure application development to production deployment across distributed computing infrastructure.

Identification	**SCONE Cross Compiler**
Release Version & Repository	**Current Version:** SCONE Cross Compiler 5.9.0 (Latest Stable) **Repository Access:** • Docker Registry: registry.scontain.com:5050/sconecuratedimages/crosscompiler • Documentation: https://sconedocs.github.io/crosscompiler/ • Azure Marketplace: SCONE Confidential Computing Platform **Key Features:** • Pre-configured development environment with all dependencies • Multi-architecture compilation targets (ARM64, x86_64) • Comprehensive language support with optimized toolchains • Enterprise-grade build artifact generation **Integration Capabilities:** • Seamless integration with SCONE runtime and sconify tool • Support for modern DevOps workflows and CI/CD pipelines • Compatible with enterprise development environments • Production-ready confidential application compilation

Table 10: SCONE Cross Compiler - Evolution and Current Capabilities

Identification	SCONE Sconify Tool
Type	A CLI tool that transforms a native Docker image into a confidential image that uses the SCONE runtime, enabling confidential computing.
Purpose	The SCONE Sconify Tool modifies existing Docker images so that applications and processes launched inside the container/image utilize the SCONE runtime, enabling execution in enclaves and benefiting from TEE support. As of 5.8–6.0, the tool supports a broader range of base images and improved integration with SCONE's policy and attestation services.
Function	The Sconify Tool relinks the SCONE libc and prepares the image for confidential execution. Key functions and updates in 5.8–6.0: - Supports multi-stage Docker builds for more efficient images. - Expanded language support: Python (3.10+), Node.js (18+), Java, Go (1.20+), C, C++. - Improved detection and handling of runtime dependencies. - Enhanced policy management: automatic policy generation and tighter CAS integration. - Improved secret injection and attestation integration. - CLI usability improvements and template-based configuration. Constraints (mostly unchanged): . All executables must be compiled with PIE (position independent code). . Original images must be musl or glibc-based. . Go support is limited to binaries compiled with gcc-go. . For Node.js, the interpreter is replaced with the SCONE curated Node.js. Additional steps: . Automatic detection and inclusion of required libraries (e.g., /usr/lib, /lib). . File system encryption and dependency scanning. . Generation of Dockerfiles, SCONE policies, and Helm charts. . Submits policies to attested CAS.
High level Architecture	The Sconify Tool is a standalone CLI tool, packaged as a Docker image. It is modularized for extensibility (plugin support added in 6.0), and communicates with SCONE CAS for policy and attestation.  native image on trustedhost SCONIFY IMAGE policy confidential imag HELM helm chat
Dependencies	Packaged as a Docker image with all dependencies included. Updated to support Docker, Podman, Buildah (5.8–6.0). Requires access to SCONE CAS for policy/attestation. Updated base libraries: OpenSSL 3.x, libc, musl.
Interfaces	Launched as a regular CLI tool on the command line. Supports declarative configuration files (YAML/JSON) for transformation policies. Improved integration with CI/CD pipelines
Data	No application data required at build time. At runtime, secrets and policies are injected via SCONE CAS.
Needed improvement	Extend support for Go-compiled programs (beyond gcc-go) and libc implementations other than GLibc and musl. (As of 6.0, Go support is still limited to gcc-go; other libc support remains a roadmap item.) Further improvements requested for custom/minimal base image handling.

Summary of the Final Release, Release Version & Repository	**Final Release Version:** See latest at `https://github.com/SCONEProject/sconify/releases` (as of 2025, version 6.0 is current). **Repository:** `https://github.com/SCONEProject/sconify` **Documentation:** `https://sconedocs.github.io/sconify/` **Release Highlights:** - Multi-stage build support, expanded language/runtime support, improved policy/attestation integration, - Enhanced CLI, CI/CD integration, modular architecture with plugin support, updated dependencies. - For detailed changelogs and migration guides, see the official documentation and release notes.

Table 11: SCONE Sconify Tool (Updated for Versions 5.8–6.0)

5.2.2 SCONE CAS (Configuration & Attestation Service)

Identification	SCONE CAS (Configuration & Attestation Service)
Type	A service that provides remote attestation as well as configuration and secret provisioning for confidential applications, running within a TEE.
Purpose	The SCONE CAS acts as a secure database and REST-based service that manages secrets, performs code attestation, and provisions configurations for confidential applications. It ensures that only authorized and attested enclaves can access sensitive data and policies.
Function	The CAS performs the following tasks, with enhancements in 5.8–6.0: . **Code Attestation:** . **Configuration Provisioning:** . **Management of SCONE Policies:** . **Key Generation & Management:** . **Secure Secret Provisioning:** . **Access Control:** . **Encrypted Code Support:** . **Isolation:** . **Audit Logging:** . **Integration with External KMS:**
High level Architecture	CAS (the Configuration and Attestation Service) is contacted by an application during its launch procedure. It verifies if the application's code and data hash (MREnclave) matches the expected measurement. Upon successful attestation, it provisions the application with configuration files and secrets (certificates, keys).
Dependencies	CAS needs to establish a connection to an Attestation Service to verify its own legitimacy and the legitimacy of client enclaves. Relies on underlying TEE hardware (e.g., Intel SGX; ARM TrustZone and AMD SEV support for client enclaves; ARM support for CAS is experimental as of 6.0). Uses updated cryptographic libraries (OpenSSL 3.x, RustCrypto) and modern container base images. Requires a secure network connection for client communication.
Interfaces	CAS provides a RESTful interface for the management of SCONE sessions and application configurations. This includes APIs for policy creation, secret upload, and attestation requests. Clients interact with CAS via secure, attested channels (e.g., TLS). gRPC APIs and OpenAPI/Swagger documentation are available for integration.
Data	CAS stores and manages SCONE policies/sessions, secrets, and configuration parameters, typically provided in YAML syntax. This data is protected within the CAS's TEE. It does not store application-specific runtime data, only configuration and secrets. Comprehensive audit logs are maintained for compliance and monitoring.
Needed improvement	Extend support for additional TEEs (e.g., AMD SEV, Intel TDX, ARM CCA) beyond Intel SGX. Further enhancements in policy language expressiveness, performance for large-scale deployments, and improved support for custom/minimal base images. Continued improvements for ARM support and cross-platform attestation.

Summary of the Final Release, Release Version & Repository	**Final Release Version:** See latest at `https://github.com/scontain/scone-cas/releases` **Repository:** `https://github.com/scontain/scone-cas` **Documentation:** `https://sconedocs.github.io/cas/` **Release Highlights:** - Enhanced policy language, improved performance for attestation and configuration management. - Modular microservices architecture, clustering, and high availability support. - Tighter integration with SCONE 6.0's expanded TEE support for client enclaves. - Robust key generation, management, and access control features within its own TEE. - Support for encrypted code and secure provisioning without source code changes. - Improved audit logging, multi-tenancy, and namespace isolation. - Integration with external KMS and enhanced CLI/SDKs for automation.

SCONE CAS (Configuration & Attestation Service) is a central policy enforcement and attestation service within the SCONE confidential computing framework. Its primary purpose is to manage and distribute secrets securely, enforce fine-grained security policies, and perform remote attestation for confidential applications running in trusted execution environments (TEEs) such as Intel SGX, Intel TDX, and AMD SEV-SNP.

Key functions of SCONE CAS include:

1. **Remote Attestation:** Verifies the integrity and authenticity of applications and their execution environments before releasing sensitive data.
2. **Policy Management:** Stores, enforces, and dynamically updates security policies that control access to secrets and configuration data.
3. **Secret Provisioning:** Securely distributes secrets (e.g. keys, certificates) only to attested and authorized enclaves, supporting secure workflows without requiring application code changes.
4. **Access Control:** Enforces strict, policy-driven access to secrets and configurations, ensuring only trusted code can access sensitive resources.
5. Audit Logging: Maintains comprehensive logs of attestation, policy changes, and secret accesses for compliance and traceability.
6. **Integration & Scalability:** Offers RESTful APIs, CLI tools, and SDKs for easy integration, and supports scalable clustered deployments for high availability.

In general, SCONE CAS acts as the trust anchor for confidential computing deployments, ensuring that sensitive data and secrets are only accessible to verified, authorised, and attested workloads, thereby enabling secure, scalable, and compliant confidential computing in cloud-native environments.

6 Performance Evaluation: Adversarial Security Experiments

6.1 TEE-based Confidential Computing for Streaming Workloads

The landscape of data streaming has undergone a significant transformation as organizations increasingly process sensitive information on distributed platforms. This evolution presents substantial security challenges, as demonstrated in financial impact assessments in which IBM states a marked increase in security breaches [9]. Organizations face average costs of $ 4.45 million when incidents occur, an increase of 15% compared to previous measurements. The increasing financial risk creates compelling incentives to implement robust security measures, particularly in systems that handle sensitive information. Conventional security approaches have addressed this challenge through layered defenses; however, critical vulnerabilities persist.

Confidential computing is emerging as the definitive solution to close security gaps by addressing limitations of software solutions through hardware-enforced trusted execution environments (TEEs) that protect data during processing [10]. This technology provides runtime protection for streaming applications through encrypted memory regions that remain inaccessible to the host operating system, hypervisor, or cloud provider. In addition, attestation mechanisms verify the integrity of the computing environment before the actual application starts. These isolation boundaries prevent unauthorized access even with administrative privileges on the host system.

The financial imperative, combined with evolving regulatory requirements for data protection, makes confidential computing increasingly essential for organizations deploying streaming applications with sensitive data workloads.

This experiment investigates the performance implications of deploying streaming clients within TEEs for streaming applications. We focus specifically on Pravega [11], a distributed streaming platform designed for high-throughput, low-latency data processing, and evaluate its performance when secured using the Secure CONtainer Environment (SCONE) framework [12], which enables applications to run inside Intel SGX enclaves [13]. Through rigorous benchmarking using the OpenMessaging Benchmark (OMB) framework [14], we compare standard Pravega clients, clients running in non-secure environments, against "sconified" clients running within TEEs across various performance dimensions.

Our work addresses a fundamental issue for secure streaming systems: it seeks to ascertain the precise latency impact of TEE's protection on streaming applications and examine the fundamental trade-off between performance and security when implementing hardware-enforced confidentiality. The proposed system is designed to preserve the low-latency characteristics essential for real-time streaming workloads.

Our experiment provides a comprehensive set of latency measurements for Pravega streaming clients, comparing standard execution (non-secured configuration) with secured execution (TEE-protected configuration) in scenarios with varying throughput.

Through detailed performance analysis of producer rates ranging from 100 to 10,000 events per second, we quantify the variation in TEE overhead with throughput intensity and determine the conditions under which performance degradation is acceptable for practical deployment. These findings show that, although TEE protection introduces significant latency at low throughput rates, its impact on performance decreases substantially as the event rate increases. Our findings enable system architects and developers to make informed decisions about implementing confidential computing solutions for high-volume streaming environments.

The remainder of this paper is structured as follows: Section 2 provides an overview of the background and related work. Section 3 describes the experiment, including the metrics and execution details. Section 4 presents and briefly analyzes the results. Section 5 concludes the study.

6.1.1 Methodology

This study relies on benchmarking a Pravega broker (a distributed streaming platform) using various configurations for both standard and secured clients. Secured clients refer to clients running in a TEE by using the SCONE runtime and network shield features (also known as Sconified clients). Standard clients are those that run without any special security configuration.

The OpenMessaging Benchmark (OMB) [14] is a vendor-neutral benchmarking framework implemented in Java and supported by the Linux Foundation. Using a Java Virtual Machine that can be run in a TEE, created by Scontain, allows for the execution of the OMB in a TEE.

Fig. 5 shows the two different types of clients used to benchmark a Pravega system. The Pravega system runs on a separate server without any specific security configuration. The sconified clients runs in a TEE and using a Network Shield, which provides encryption and an application-level firewall that denies untrusted connections.

Further, different configurations are considered to have a better understanding of the impact of sconified benchmarks:

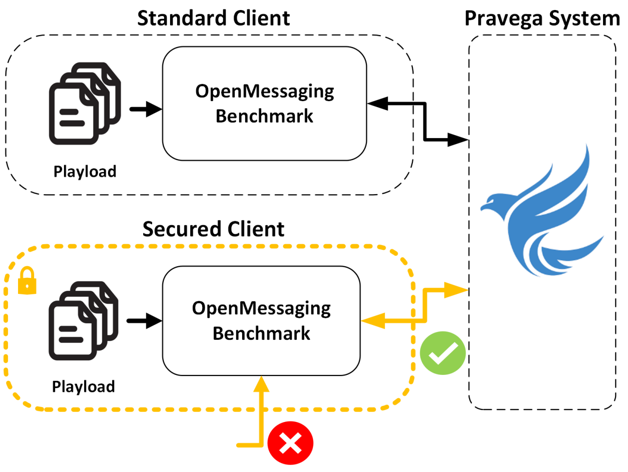


Figure 5: Experiment workflow: on the left are the standard and secured (Scone) benchmarking clients, and on the right there is the Pravega system. Each side is running on a different server. The clients are running in containers, and the secured one includes Scone's features such as the network shield.

- **Production rate:** 100, 1000 and 10000 events per second.
- **Security:** Standard and Secured.
- **Event size:** 100B, 1KB, 10KB and 100KB. Payload file sizes are generated from the default payload files in the OpenMessaging Benchmark tool.

Permuting the depicted scenarios and configurations produces 24 total test cases considered in this study. Each test case is configured with a 1-minute warmup and 5 minutes of benchmarking. The number of event or messages sent varies depending on the production rate.

Naturally, The OpenMessaging Benchmarking tool produces results that slightly vary for repetitions of the same test and rarely it produces outliers. We execute each test case 10 times and report the averages to address variations and outliers in the results.

This section presents a list and brief descriptions of the metrics employed in this experiment. Most of these metrics are derived from the OpenMessaging Benchmark tool, which generates a JSON-formatted output file containing various measured and computed variables collected during the benchmarking process.

- **Write latency at 50% or 50 percentiles:** Publisher write latency at the 50 percentiles (median). Variable *aggregatedPublishLatency50pct*.
- **Write latency at 75% or 75 percentiles:** Publisher write latency at the 75 percentiles. Variable *aggregatedPublishLatency75pct*.
- **Write latency at 95% or 95 percentiles:** Publisher write latency at the 95 percentiles. Variable *aggregatedPublishLatency95pct*.
- **Write latency at 99% or 99 percentiles:** Publisher write latency at the 99 percentiles. Variable *aggregatedPublishLatency99pct*.
- **Cumulative distribution function of latencies:** A list of values produced of applying the cumulative distribution function to all the latencies of the benchmark. Variable *aggregatedPublishLatencyQuantiles*.
- **Throughout:** Total output data divided by the total execution time in MB per second. [16]

KIO Networks [15], acting as the infrastructure provider in the Neardata European research project, supplied the computing infrastructure for the technical experiments. The infrastructure layer consists of three servers with the following hardware specifications: Intel® Xeon® Platinum 8458P processors (based on the "Sapphire Rapids" architecture), 1 TB of RAM, and 20 TB of storage. The virtualization layer hosts the virtual machines required for the whole project experiments. Fig. 6 illustrates this two-tier architecture infrastructure.

[16]It was calculated using the total bytes (obtained from a modified version of the OpenMessaging Benchmark) and the execution time (by resting timestamp before and after the benchmark run, excluding the warm-up period).

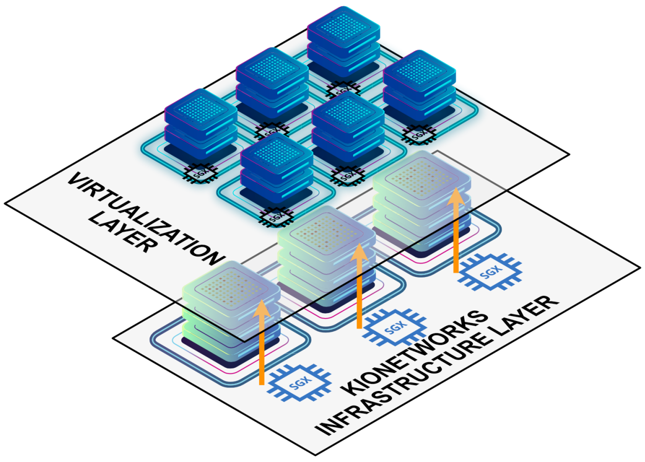


Figure 6: Layered SGX-enable architecture for secure virtualization.

For this experiment, two virtual machines were needed. Table 13 presents the hardware specifications of these Virtual Machines (VMs). Both VMs run the Linux operating system, and VM 1 includes an Intel SGX hardware and drivers.

Table 13: Virtual Machines hardware details

	VM 1	**VM 2**
Services	Docker and Clients	Pravega
Memory	32G	64G
Processor	Intel Xeon Platinum 8458P	Intel Xeon Platinum 8458P
Cores	8	16
Features	Intel SGX	-
Disk	50G	300G
Extra Disks	-	2 SSD: 250G each

The Pravega service deployed in VM 2 is using a standard configuration (no security or encrypting configuration): one Bookkeeper, one Zookeeper, a Pravega controller and segment store that is connected to a external S3 external storage service.

6.1.2 Benchmarking Performance in TEEs and Standard Environments

The main objective of this study is to evaluate the performance of streaming clients operating in both TEEs and standard environments. Specifically, we aim to identify the configuration that yields the lowest latency under various test conditions. As the results demonstrate, the answer depends heavily on the scenario.

Figures 7 and 8 present the cumulative distribution function (CDF) of latencies plots for test cases with producer rates of 100 and 1,000 events per second, respectively. In both cases, standard clients consistently demonstrate lower latency than secure clients, around 50% less latency (e.g. in producer rate 100 and event size 1KB the average write latency at 99% is 3.632ms for non-security and 9.373ms for security environments). Although the performance gap is evident, both figures reveal a trend: larger event sizes are associated with a smaller disparity in latencies between the two configurations (e.g. in producer rate 1,000 and event size 100KB the average write latency at 99% is 59.692ms for non-security and 71.967ms for security environments).

Figure 9 extends this analysis to a higher producer rate (10,000 events per second). It exhibits a similar pattern, namely, the latency gap between standard and secure clients narrows as the data rate increases (e.g. in producer rate 10,000 and event size 10KB the average write latency at 99% is 69.577ms for non-security and 78.465ms for security environments).

Notably, in figure 9 in the subplot where the event size is 100KB, the secure client actually achieves lower latency that its standard counterpart, for the average write latency at 99% is 15,450.847ms for the standard

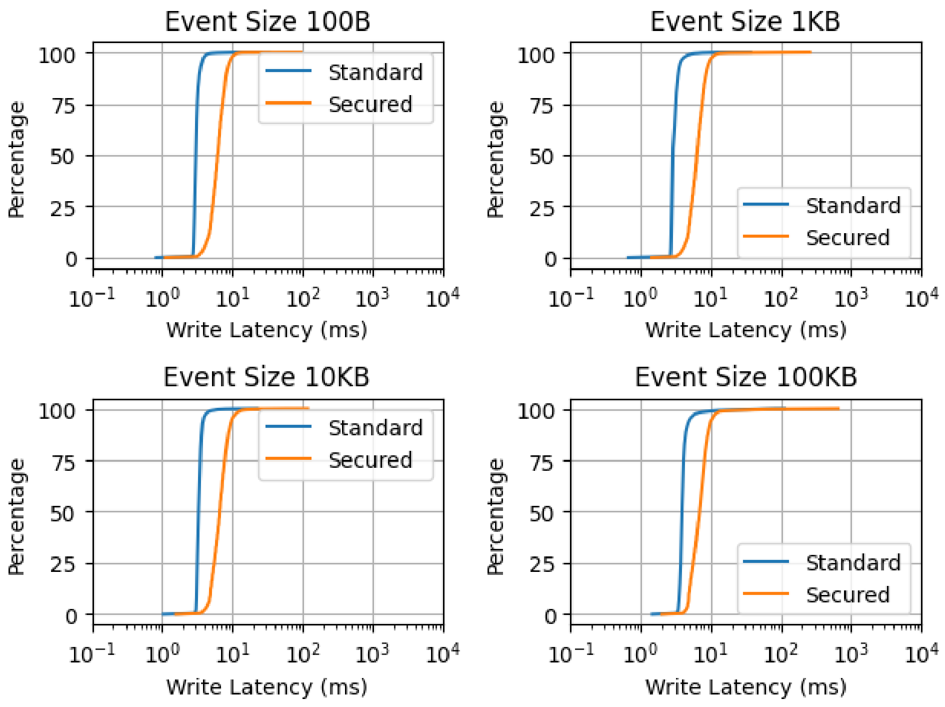


Figure 7: Cumulative distribution function of logarithmically scaled write latencies for test cases with a producer rate of 100 events per second across varying event sizes. The plots indicate that standard clients exhibit lower latencies compared to secure clients.

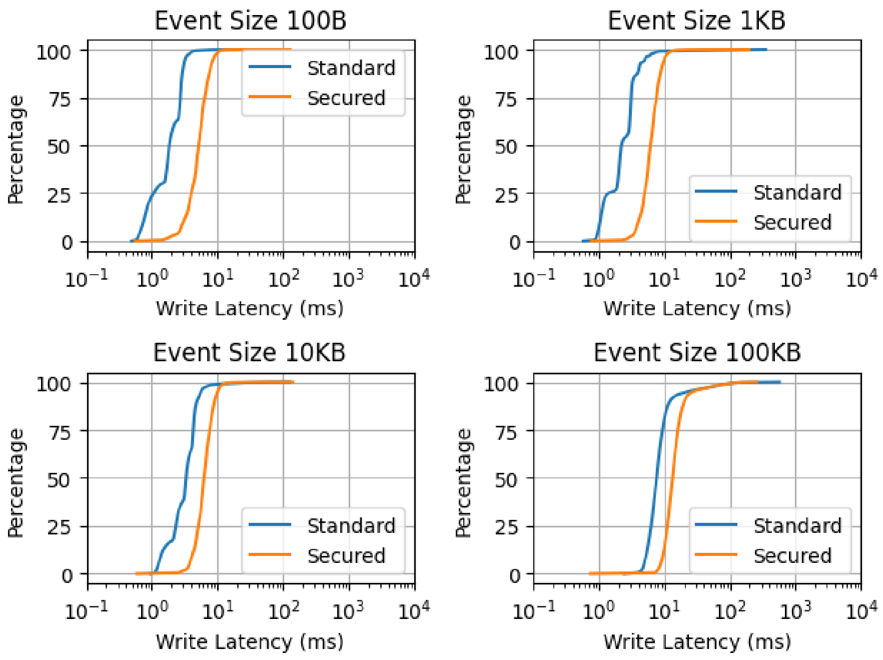


Figure 8: Cumulative distribution function of logarithmically scaled write latencies for test cases with a producer rate of 1,000 events per second across varying event sizes. The plots indicate that standard clients exhibit lower latencies compared to secure clients.

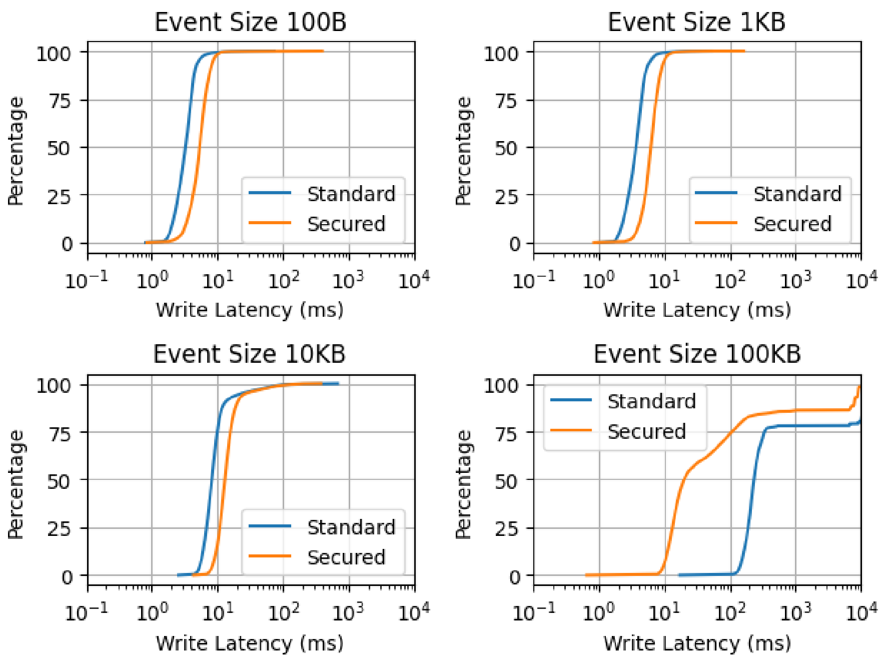


Figure 9: Cumulative distribution functions of logarithmically scaled write latencies for test cases with a producer rate of 10,000 events per second across varying event sizes. The plots indicate that for event sizes of 100B, 1KB, and 10KB, standard clients exhibit lower latencies compared to secure clients. However, for the 100KB event size, secure clients outperform standard ones in terms of latency.

and 9,632.655ms for the secure environment. This particular test case stands in contrast to the other ones and suggests that the service may have reached its maximum data throughput capacity under these conditions.

6.1.3 Scenarios Where Secure Clients Outperform

Figure 10 supports the hypothesis outlined in the previous section. Benchmark results for a 100 KB event size at a producer rate of 10,000 events per second indicate the maximum throughput that the deployed Pravega service can sustain with the current hardware configuration. While the first three plots illustrate increasing data throughput, driven by larger event sizes and higher producer rates, the final plot shows an unexpected outcome: when using 100 KB events, the system transmits less data at 10,000 events per second than it does at 1,000 events per second.

This anomaly suggests a saturation point in the service's capacity. Interestingly, under these high-load conditions, the secured client demonstrates superior performance compared to the standard client, contrary to earlier findings. According to the Scone development team, this can be attributed to Scone's custom implementations for multi-threading, busy looping, and memory management. These optimizations occasionally allow sconified applications to outperform their vanilla counterparts [16, 17].

6.1.4 Latency Differences Between Standard and Secured Environments

Our findings demonstrate that secure clients generally experience higher write latency compared to their standard counterparts. Quantifying this difference is essential to assess the scenarios in which performance degradation is acceptable, given the security benefits. To this end, we compute the percentage difference in write latency between standard and secured clients.

Figure 11 visualizes these ratios. In the first subplot, representing a producer rate of 100 events per second, the average standard write latencies range from 40% to 60% of those observed with secured clients. As both the producer rate and event size increase, the latency gap widens to approximately 40% to 80% as shown in the second subplot. In the final subplot, this divergence continues until the Pravega system reaches its maximum throughput capacity, at which point the results become unreliable.

Executing Pravega clients within a TEE introduces a measurable write latency overhead compared to standard execution. At a producer rate of 100 events per second, standard clients latency represents around 50% of the secure clients latency (secure clients latency is around the double of the standard clients latency). When

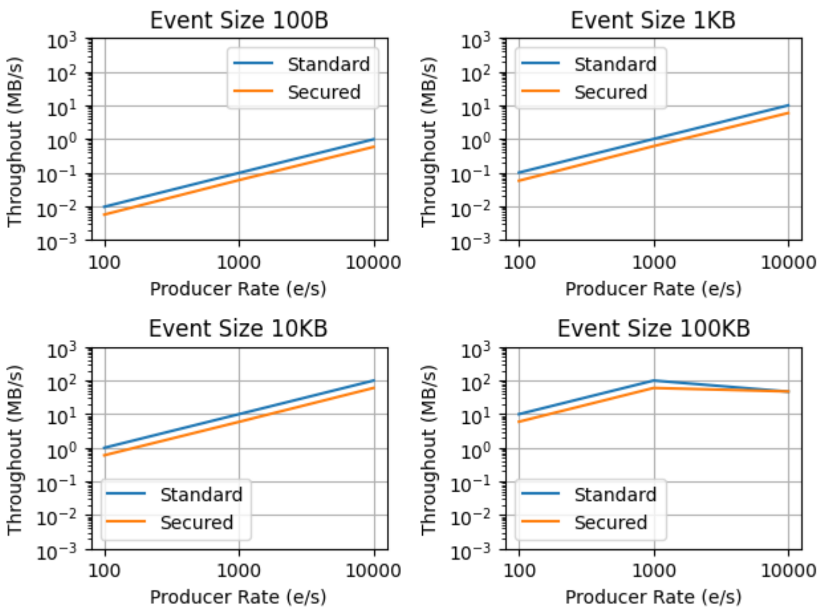


Figure 10: Logarithmic scaled throughput (MB per second) results from standard and secured OMB clients using different producer rates (events per second) and event sizes. The plot shows that standard clients manage to send more data than the secured ones. The last test case had an exception (event size of 100 KB and producer rate of 10,000 e/s). Standard and secure clients produce similar throughput, but it seems like there is a saturation point in the service's capacity.

the rate increases to 1,000 events per second, the standard client's latency ranges between 20% and 60% of its secure counterpart's latency. At the highest tested rate of 10,000 events per second, the performance gap further narrows.

These findings are essential for identifying scenarios where the trade-offs of running clients in a TEE, such as those facilitated by SCONE, are justifiable. One such promising application is the Surgomics use case from the Neardata European research project. This scenario involves supporting surgeons with real-time video analytics during operations, where both low latency and high data throughput are critical.

Given the highly sensitive nature of medical data, security becomes paramount, even at the expense of some performance loss. Our results suggest that, in such high-throughput environments, the performance degradation introduced by secure execution is limited and may be acceptable when balanced against the benefits of enhanced data protection.

Future research could validate these findings by evaluating other distributed streaming platforms, such as Apache Kafka and Apache Pulsar, to ensure our conclusions are not specific to this particular implemen-

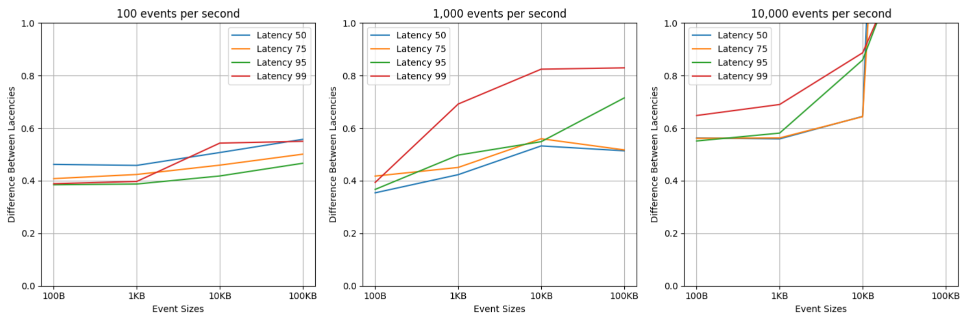


Figure 11: Percentile difference between standard and secured latencies. At 100 e/s, the difference ranges from 40% to 60% . At 1,000 e/s, it increases to 40%–80% . At 10,000 e/s, the difference remains similar for small event sizes but rises sharply when Pravega reaches its throughput limit.

tation. A critical extension would involve Sconified the Pravega server-side components. This investigation would encompass a comprehensive benchmark assessment of the Sconified Pravega server architecture and a detailed performance comparison between native and TEE-protected server components. Such an analysis would provide a comprehensive understanding of the I/O performance implications of deploying SCONE across both the client and server sides of the Pravega streaming platform.

6.2 Experimental Validation of TEE Security Mechanisms-SinClave: Hardware-assisted Singletons for TEE Attestation

This section presents experimental evaluation of advanced TEE security mechanisms critical for **NEARDATA Objective O-3: Provide secure data orchestration, transfer, processing and access**. The experiments validate security enhancements required for the Data Broker service to enable trustworthy data sharing and confidential orchestration of data pipelines across the Compute Continuum. The evaluation specifically addresses **KPI-4: High levels of data security and confidential computing validated using TEEs and federated learning in adversarial security experiments**.

The research addresses fundamental security vulnerabilities in TEE-based distributed systems that directly impact the Data Broker's ability to provide secure data orchestration. Four critical attack vectors are examined: remote attestation bypass attacks exploiting lack of completeness and freshness in measurement processes, storage rollback attacks targeting persistent state during container restarts, instantiation attacks where adversaries spawn multiple service instances, and end-user trust validation failures in distributed web-facing services.

These vulnerabilities are particularly critical for federated scenarios where surgical AI systems require trusted collaboration between hospital networks, and metabolomics research platforms need secure data sharing across pharmaceutical and academic institutions. The Data Broker must ensure comprehensive protection across runtime attestation, persistent storage, instance control, and end-user validation to meet the stringent security requirements of healthcare and research applications.

Current remote attestation protocols in TEEs suffer from two critical limitations that undermine federated learning trust establishment: (1) *Completeness* - failure to measure all aspects influencing application behavior, and (2) *Freshness* - verification only of initial enclave state without detecting runtime modifications. These vulnerabilities enable "reuse attacks" where adversaries manipulate enclaves to impersonate legitimate instances while bypassing attestation protocols.

Comprehensive analysis revealed vulnerabilities across multiple TEE frameworks including SCONE, SGX-LKL, and Occlum. These frameworks only verify enclave initial state, making them incapable of detecting dynamically loaded malicious libraries or manipulated configurations that can alter application behavior after attestation. The fundamental vulnerability lies in the gap between initial measurement and runtime behavior.

Attack Implementation Results: The attack against SCONE-based enclaves successfully exploited the fact that program file systems are not reflected in enclave measurements. Python applications using identical interpreters produced indistinguishable enclave measurements, allowing adversaries to substitute malicious Python programs. The TEE impersonator required only 75 lines of code and successfully obtained user configurations by presenting valid SGX reports from the compromised enclave.

The SGX-LKL framework attack leveraged encrypted disk images where attestation occurs before user code loading. Two different programs running in SGX-LKL appeared identical from SGX attestation perspective. The attack required 236 lines of Python code for the impersonator and 152 lines of C code for the report server, demonstrating practical exploitability across multiple TEE frameworks with minimal implementation effort.

6.2.1 SinClave Protection Mechanism

SinClave introduces a hardware-assisted mechanism that enforces both freshness and completeness in TEE remote attestation through Singleton Enclaves. This represents the first practical defense against reuse attacks in TEE frameworks, addressing a previously overlooked vulnerability in remote attestation protocols critical for establishing trust in federated computing environments.

Attestation Token Innovation: Each enclave instance is uniquely individualized through a cryptographic attestation token generated by the verifier and embedded during enclave creation. This token ensures that only one instance of each enclave can be attested, preventing reuse or impersonation attacks that exploit identical enclave measurements.

Modified Measurement Process: The solution extends the SGX measurement process by incorporating an instance page containing the attestation token and verifier cryptographic identity. This page is hashed into the enclave's MRENCLAVE measurement during construction, enabling verifiers to detect runtime modifications or reuse attempts. Unlike previous static configuration approaches, SinClave introduces on-demand SigStruct creation that maintains both security and flexibility.

6.2.2 Performance Evaluation Results

Cryptographic Performance Impact: SHA-256 computation performance comparison demonstrates the trade-off between security enhancements and computational throughput. The Ring library baseline achieves approximately 405 MB/s throughput across all buffer sizes. The SinClave implementation yields about 180 MB/s, representing a 2.25× overhead due to less optimized code implementation. The SinClave-BaseHash variant shows equivalent performance to the standard implementation for large buffers, but performs better for small buffers (245 MB/s for 2KB buffers).

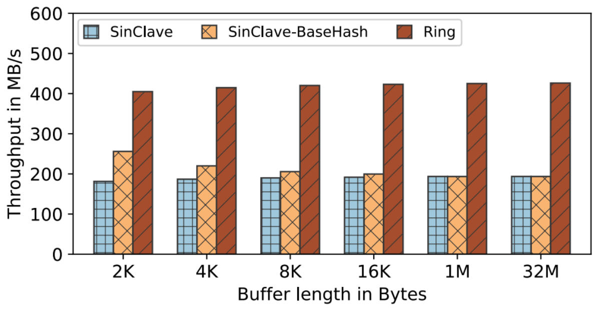


Figure 12: Calculation of a SHA256 checksum with different implementations

SinClave's secure, interruptible SHA-256 implementation introduces a 2.25× overhead compared to the highly optimized Ring library. This is expected due to additional security checks and the need for interruptibility in TEE contexts. For small buffer sizes, SinClave-BaseHash outperforms the standard implementation, indicating optimization for certain use cases

Compilation Overhead Analysis: Compilation duration results show that native compilation requires only 0.033 seconds, while baseline SCONE needs 1.52 seconds, and SinClave requires 6.26 seconds. The 4× increase over baseline reflects the interruptible SHA-256 implementation overhead during iterative hash computation. SigStruct signing operations take 4.9 ms on average, with verification requiring 0.4 ms. The singleton page retrieval process adds 26.3 ms to enclave startup, with the majority of time spent on database access and policy enforcement operations.

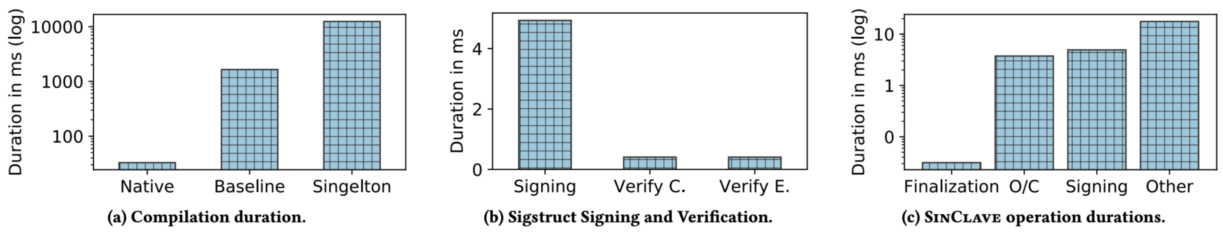


Figure 13: Micro-benchmarks of SinClave.

Application Performance Impact: Real-world application evaluation reveals that SinClave introduces minimal overhead for practical applications. Python applications show 1.03% overhead (2.43s baseline vs 2.46s with SinClave), OpenVino image classification demonstrates 2.49% overhead (45.2s vs 46.3s), and PyTorch CIFAR-10 training exhibits 13.2% overhead (128.7s vs 145.7s). The PyTorch result represents the highest overhead case, which remains acceptable for applications requiring strong confidentiality guarantees.

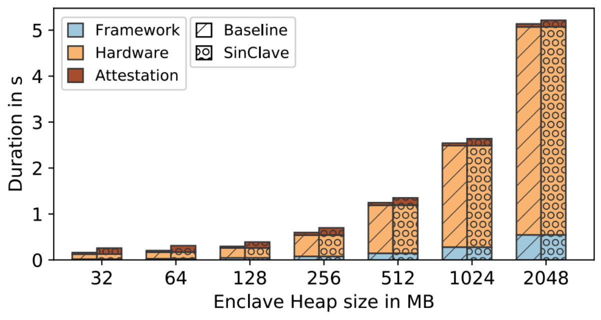


Figure 14: Measurement of program execution.

Execution Overhead Analysis: For attested execution modes, the protection mechanism added 132–144.2 ms compared to 36.3–65.9 ms for baseline systems. The overhead variation correlates with enclave heap size due to cache pressure effects, but remains acceptable relative to total execution times ranging from 258.1 ms to 5.2 seconds. For non-attested execution modes, SinClave showed overhead within single-digit microsecond values, confirming that security enhancements do not significantly impact non-security-critical operations.

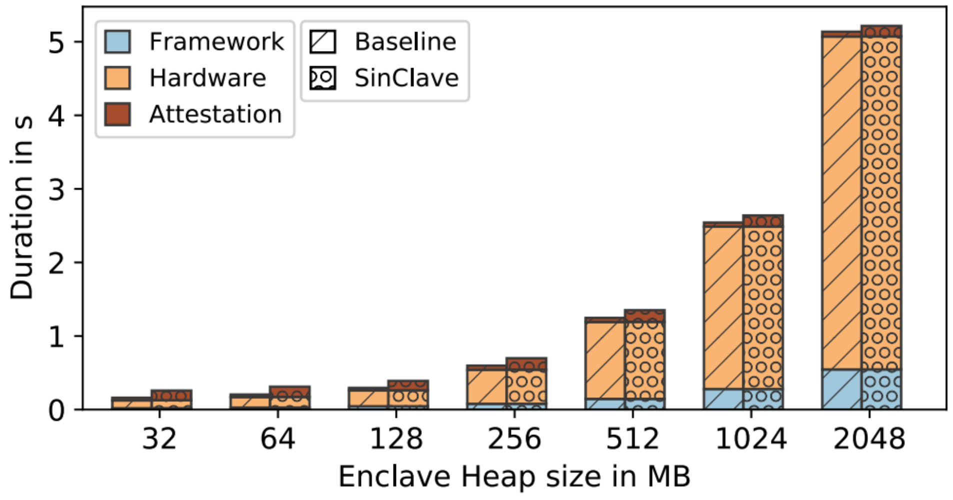


Figure 15: The performance overhead of SinClave with real-world workloads.

For general-purpose and AI inference workloads (Python, OpenVINO), SinClave introduces minimal overhead (1–2.5For compute- and memory-intensive machine learning training (PyTorch), the overhead is higher (13.2%) but remains within the range observed for comparable TEE solutions, especially given SinClave's stronger security guarantees

SinClave's experimental validation is fully aligned with NEARDATA's performance evaluation requirements. It demonstrates a strong security–performance tradeoff by providing robust guarantees of freshness, completeness, and singleton enforcement with minimal overhead, meeting KPI-4. The evaluation is conducted on representative workloads from domains central to NEARDATA's mission, including healthcare, federated learning, and AI pipelines. Following industry best practices, the study reports both percentage and absolute overheads, details the hardware and software environment, and benchmarks performance at both micro- and macro-levels, consistent with NIST and cloud security guidelines. In comparative context, SinClave achieves overheads of 1.03%–13.2%, which are competitive with or superior to state-of-the-art TEE-based security solutions, while offering a stronger security model through its unique singleton enforcement and comprehensive attestation.

6.3 CRISP: Confidentiality, Rollback, and Integrity Storage Protection

6.3.1 Storage Rollback Protection Architecture

CRISP addresses the critical gap where TEE integrity protection does not extend to persistent storage, essential for federated learning scenarios where frequent container restarts create opportunities for state manipulation attacks. The Data Broker's storage protection addresses rollback attacks where adversaries can exploit the disconnect between TEE memory protection and storage persistence to undermine collaborative trust relationships.

Rollback Attack Threat Model: In federated environments supporting distributed data orchestration, adversaries can access sensitive surgical training data or metabolomics research results, then roll back audit logs to conceal unauthorized access patterns. Malicious actors within multi-institutional collaborations could circumvent recently revoked access permissions by reverting authorization databases to previous states, effectively re-enabling terminated access rights. For surgical AI federated learning, attackers could roll back model training checkpoints to earlier states, potentially degrading collective intelligence while concealing manipulation.

Monotonic Counter Integration: CRISP leverages monotonic counters to bind version information with write operations through the SCONE runtime, providing cryptographically verifiable storage freshness guarantees. The mechanism operates transparently within data orchestration pipelines, incrementing counters on critical disk synchronization operations including explicit fsync calls, file closure events, and program termination sequences. The integration embeds counter values directly in storage metadata managed by SCONE's File System Protection Framework (FSPF).

When Data Broker services restart, the SCONE runtime performs automatic integrity verification by comparing stored counter values against current monotonic counter state. Any discrepancy indicates potential rollback manipulation, causing the Data Broker to halt operations and require administrative intervention. This approach provides local rollback detection without requiring coordination with remote trusted parties, reducing network dependencies in federated environments.

6.3.2 Optimistic Batching Performance Enhancement

To address performance overhead of frequent counter operations that could impact real-time federated learning workflows, CRISP introduces optimistic batching that consolidates multiple write operations into single counter increments. This mechanism is crucial for applications where surgical AI systems require low-latency data sharing between hospitals, and metabolomics platforms process high-throughput analytical data streams.

The batching mechanism employs a dedicated mc-thread that processes accumulated operations asynchronously, enabling the Data Broker to acknowledge write requests immediately without blocking federated learning workflows on counter update latencies. The approach introduces a controlled vulnerability window during batch processing, where recent writes may not yet be rollback-protected. However, this window can be precisely configured based on application requirements through tunable batch parameters.

6.3.3 Performance Evaluation Results

Raw Disk Performance Analysis: CRISP-protected writes often outperform native execution due to caching mechanisms in optimistic batching. Write operations are cached in memory-mapped regions and flushed in larger chunks, reducing context switching overhead and masking encryption costs. For buffer sizes up to 16KB, CRISP maintains competitive throughput with baseline SCONE implementations.

Reading performance shows expected 2–3× overhead patterns due to CRISP requiring access to multiple files (data, vault metadata, FSPF metadata) and decryption operations. This overhead reflects the security-performance trade-off for ensuring data integrity and freshness verification on every read operation.

Raw Disk Performance Analysis

- **Write Performance:** CRISP-protected writes often outperform native execution for small buffer sizes (up to 16KB). This is due to caching mechanisms in optimistic batching, where write operations are cached in memory-mapped regions and then flushed in larger chunks. This approach reduces context switching overhead and masks the cost of encryption.
  - **Native:** Baseline performance.
  - **HW:** SCONE runtime with Intel SGX hardware protection.
  - **HW+FSPF:** SCONE with filesystem protection (FSPF).
  - **MC:** CRISP with default monotonic counter configuration.
  - **MC+Rate Limit:** CRISP with a 100 ms monotonic counter rate limit.

Key observation: CRISP (MC) outperforms HW+FSPF and is competitive with native execution for buffer sizes up to 16KB. For the largest buffer size (32KB), native execution is fastest, but CRISP remains close.

- **Read Performance:** Reading performance shows a 2–3× overhead compared to native execution. This is because CRISP requires access to multiple files (data, vault metadata, FSPF metadata) and must perform decryption operations. 16demonstrates this overhead, showing that both HW+FSPF and CRISP configurations have significantly lower throughput than native execution. *Key observation:* The overhead results directly from the security–performance trade-off: every read operation must verify data integrity and freshness, which involves additional I/O and cryptographic checks.

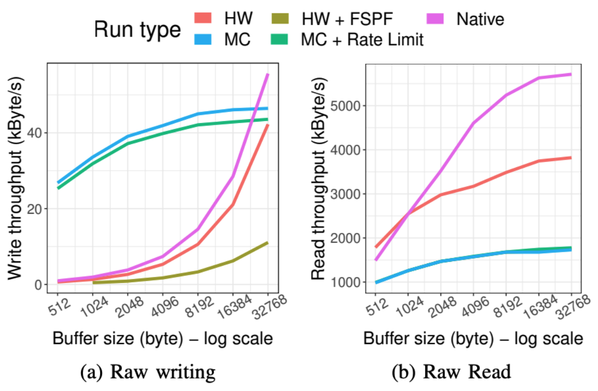


Figure 16: Raw disk performance experiment

Vulnerability Window Control Analysis:

- **Checker API Functionality:** The Checker API provides blocking mechanisms to ensure that critical writes are protected by a monotonic counter before applications externalize dependent information.
- **Performance Impact 17:**
  - With 1% checker probability, applications experience 31% average throughput compared to unprotected variants, while achieving 77% of native throughput.
  - With 10% checker probability, single-threaded applications show performance below encrypted filesystem variants, indicating the need for careful tuning based on security requirements.
- **Key Observation (Table II):** Table II reports the average number of `fsync` calls per batch and the average batch duration. Increasing the checker probability reduces the vulnerability window but also decreases throughput, highlighting the trade-off between security and performance.

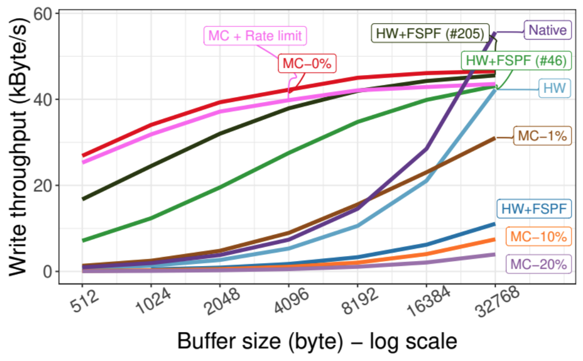
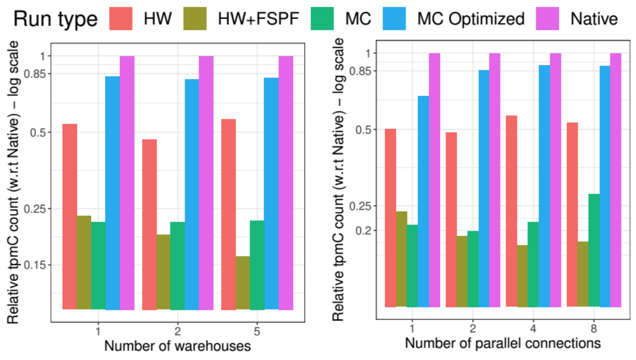


Figure 17: Raw write throughput with various configurations

Production Database Evaluation (MariaDB TPC-C Benchmark)

- **TPC-C Benchmark:** The TPC-C benchmark is a standard for evaluating transactional database performance. It simulates a realistic OLTP workload with a mix of read and write operations and scales with the number of warehouses and connections.
- **Performance Results 18:**
  - The optimized CRISP configuration achieves 0.83× native performance while providing comprehensive rollback protection.
  - The exclusive `mc-thread` design allows immediate write acknowledgment and processes rollback protection asynchronously, preventing connection threads from blocking on counter operations.
- **Scalability:** As workload complexity increases (e.g., more warehouses and connections), CRISP maintains consistent scaling characteristics, demonstrating its suitability for high-concurrency, stateful applications.



(a) Relative tpmC throughput with varying warehouses

(b) Relative tpmC throughput with varying connections

Figure 18: MariaDB TPC-C experiment

Identity Management System Performance (SPIRE Evaluation)

- **SPIRE Evaluation:** SPIRE (SPIFFE Runtime Environment) was evaluated for certificate signing operations across 25 agents managing 9000 SVID entries each.
- **Performance Results 19:** With 20% Checker API probability, the system maintains stable throughput without significant degradation.
- **Resource Utilization:** CPU utilization patterns remain consistent, indicating that rollback protection overhead does not create resource bottlenecks for typical identity provisioning scenarios.

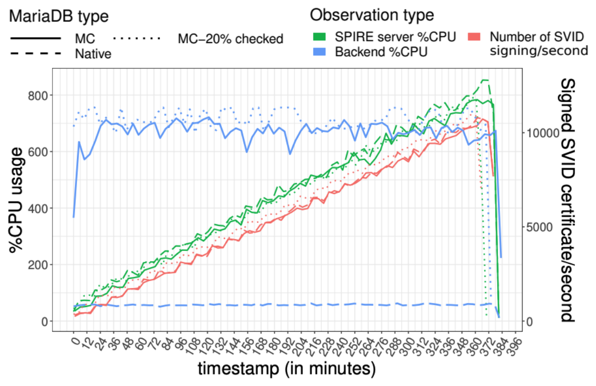


Figure 19: : Certificate signing throughput with CPU load on the SPIRE server and MariaDB backend

Multi-Server Scalability Analysis

- **Scalability with Multiple Servers (Figure 8):** 20 demonstrates the scalability of multiple TEE-equipped SPIRE servers sharing a single CRISP-enabled MariaDB backend.
- **Performance Results:** With 10 servers, the average overhead compared to a single native server deployment is 25%, confirming that storage protection does not become a scaling bottleneck for distributed architectures.
- **Scaling Characteristics:** The system scales well up to 10 servers, after which performance plateaus, indicating the backend's maximum capacity for the given workload.

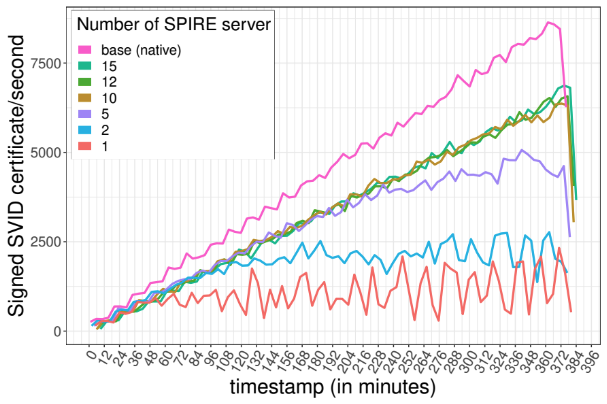


Figure 20: Certificate signing throughput on one or more trusted SPIRE servers

The figures and data from the evaluation demonstrate that CRISP provides strong rollback protection and data security with minimal performance penalties in most scenarios. Its design, including optimistic batching and asynchronous counter operations, allows it to outperform or match baseline secure implementations (like SCONE with FSPF) in write-heavy workloads, while maintaining acceptable overheads for read operations and scaling effectively in distributed, multi-server environments.

6.4 Last-Level Defense for Instantiation Attack Tolerance

6.4.1 LLD Architecture for Data Orchestration

Building upon the Last-Level Defense (LLD) architecture for application integrity and confidentiality, we present an enhanced implementation that addresses critical instantiation attacks in distributed data processing environments supporting NEARDATA Objective O-3. LLD's TEE-supported leases, rollback detection through monotonic counters, and transparent runtime protection provide essential security mechanisms for enabling trustworthy data sharing and confidential orchestration of data pipelines, specifically addressing KPI-4 through high levels of data security and confidential computing validated using TEEs and federated learning in adversarial security experiments.

Instantiation Attack Threat Model: In compromised environments, attackers can force illegal executions and start additional instances to interfere with replication and agreement protocols, leading to inconsistencies in replicated data or routing clients maliciously to spurious service instances. Attackers can modify time perception to use expired authentication codes or certificates, and roll back storage to circumvent access revocations and undo undesired actions.

The LLD Runtime ensures that enclave applications cannot alter their states without authorization by intercepting system calls and verifying execution leases periodically. The LLD Lease Service ensures applications work as single-instance services, with at most one instance running at any time. The core component is the LLD Store, an integrity- and rollback-protected replicated state database that keeps leases and guarantees their integrity and durability.

6.4.2 Instantiation Control Mechanisms

LLD Runtime Protection: The enhanced LLD Runtime implementation extends SCONE's SGX framework to provide transparent protection for data processing components across distributed environments. Each data orchestration workflow component operates within individual TEEs, with the LLD Runtime automatically enforcing instantiation control through verified execution leases.

The runtime intercepts system calls that may externalize enclave state, performing lease validation before allowing data processing operations to proceed. This approach prevents Sybil and Fork attacks where adversaries spawn multiple instances of data processing services to manipulate aggregation results or compromise federated learning consensus protocols. The runtime executes every 1 second or whenever a system call that may externalize internal enclave state occurs.

Trusted Lease Service: The LLD Lease Service manages execution permissions for data processing instances across distributed infrastructure. The service maintains cached lease states for performance optimization while ensuring consistency through conditional database transactions that prevent lease conflicts between multiple service instances. Lease requests comprise an instance ID and timestamp from a trusted time source, representing when the instance makes the request.

Local Leader Election Protocol: The Local Leader Election protocol ensures single-instance operation for critical infrastructure components using monotonic counter-based election mechanisms. This prevents instantiation attacks on core data orchestration services while supporting automatic recovery from component failures. Each monotonic counter has a unique, non-clonable ID, with candidates attempting to increment counter values using their unique instance identifiers.

6.4.3 Performance Evaluation Results

- **Introduction to LLD:** LLD (Last-Level Defense) is a multi-layered, TEE-augmented framework that protects applications from Sybil, Fork, and Rollback Attacks—even when adversaries fully control the host, cloud, or orchestration layer. It leverages secure leases, trusted time, monotonic counters, and a rollback-protected distributed data store to ensure:
  - Only one legitimate application instance operates at a time (instantiation/fork attack resistance).
  - Past states cannot be replayed (rollback resistance).
  - Confidentiality and integrity are preserved even for unmodified, off-the-shelf applications.
- **Experimental Setup:**
  - **LLD Store:** Dqlite running on 3 Intel NUCs with SGXv2 (hardware monotonic counters).
  - **LLD Lease/Clock Server:** High-end server with Icelake-SP processors.
  - **Client:** Runs atop LLD.
  - Monotonic counter latency $\approx$ 40ms per verified write.

- **LLD Store: Rollback and Instantiation Attack Tolerance**
  - **Startup Delay and Leadership Check Overhead Figure 21:**
    - Measured the overhead of enforcing single-instance leadership with periodic checks (interval P).
    - Increasing P (less frequent checks) only marginally affects startup time or work lost.
    - **Optimal configuration:** $P = 1s$ offers best crash recovery with negligible extra overhead.

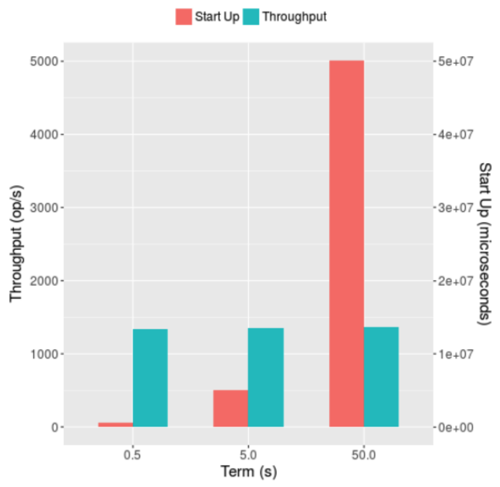


Figure 21: Start-up delay and work lost per period in 3 configurations

  - **Local Write Performance with Security Layers (Figure 22):**
    - Tested local writes (64B and 8KB) under four protection scenarios: 0. No protection, 1. TEE-only integrity, 2. TEE + encrypted/integrity-protected storage, 3. Full LLD (TEE + encryption + rollback protection with monotonic counters).
    - Findings:
      - Monotonic counters dominate cost for small writes (∼40ms/op).
      - Encryption adds ∼25% overhead for small writes, but is masked for large writes due to filesystem buffering.
      - For large writes, encryption overhead is negligible compared to monotonic counters.
    - Interpretation: While monotonic counters slow frequent small writes, real-world workloads with batched or larger operations see much less impact.

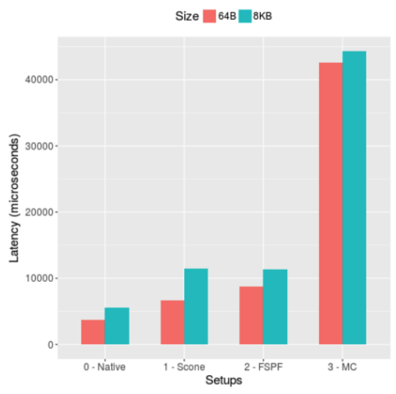


Figure 22: Cost of LLD writes with different levels of security

- **Protecting Off-the-Shelf Services**

- **Dqlite Cluster Benchmark (Figure 23):**
  - 3-node Dqlite cluster, workloads: read-only, 5% write, write-only.
  - Configurations: none, TEE, TEE+encryption, full LLD.
  - Findings:
    - TEE and TEE+encryption show similar performance.
    - Full LLD incurs 43–49% performance loss due to monotonic counter updates.
    - However, not every syscall triggers a lease update, so higher-level workloads see less impact.

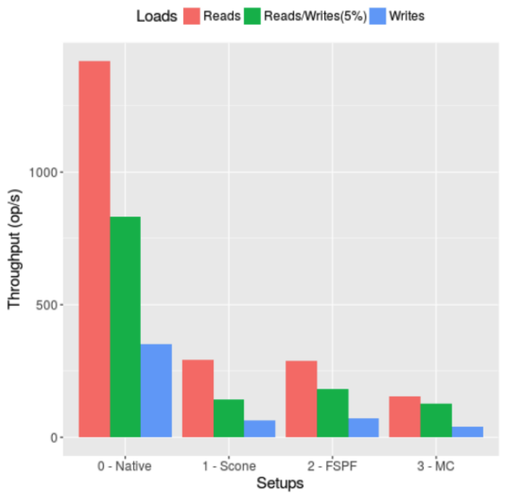


Figure 23: Performance of the Dqlite 3-node cluster with increasing layers of protection measures.

- **MariaDB TPC-C Benchmark (Figure ??):**
  - MariaDB server with TPC-C, five configurations: none, regular TEE, TEE-tuned, TEE-tuned+encryption, full LLD.
  - Findings:
    - With tuning (e.g., adjusting thread sleeps/spins, caching), full LLD adds < 12% overhead.
    - For I/O-intensive but high-level applications, LLD overhead is well amortized.

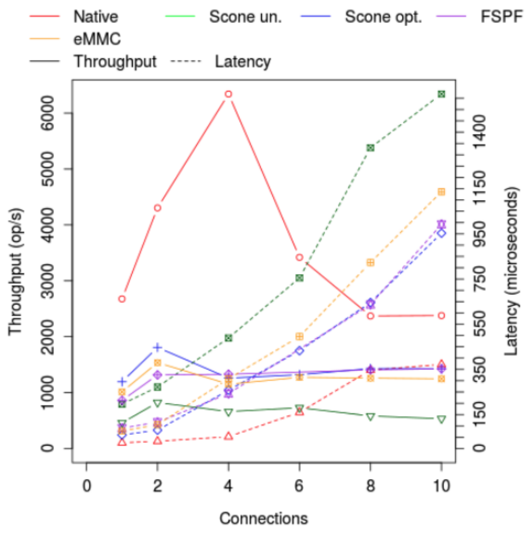


Figure 24: : Performance on TPC-C of MariaDB with increasing levels of security

This paper presents LLD (Last-Level Defense), a practical and transparent solution to enhance the integrity and confidentiality of applications running in untrusted environments—such as public clouds—by extending the guarantees provided by hardware Trusted Execution Environments (TEEs) to resist Sybil, Fork, and Rollback Attacks. LLD introduces a novel combination of execution leases, trusted time, and monotonic counters, enforced through a distributed and rollback-protected database, to ensure that only a single instance of a trusted application can operate at any time, and that its state remains consistent and fresh. Crucially, LLD achieves this protection without requiring changes to application code or sacrificing compatibility with standard cloud orchestration tools, making it suitable for deployment on unmodified, off-the-shelf services and databases. Performance evaluations demonstrate that, while the use of monotonic counters and trusted lease checks introduces some overhead—particularly for high-frequency, fine-grained operations—these costs are amortized in realistic workloads, with less than 12% overhead for complex, optimized applications like MariaDB under TPC-C benchmarks. Aligned with the NearData project, LLD empowers organizations to securely process and manage sensitive data close to where it is generated, even when the surrounding infrastructure cannot be trusted. By ensuring that data remain protected against powerful adversaries and that application integrity is verifiable, LLD supports NearData's vision for secure, reliable and resilient edge and cloud computing. This work demonstrates that with careful systems design, it is possible to bridge the gap between strong theoretical security guarantees and practical, scalable, and transparent deployment in real-world environments.

7 Conclusions

The NEARDATA framework has made substantial advancements in its security architecture over the past year, with a particular emphasis on the NEARDATA Data Broker and its integration with key security tools such as the SCONE runtime, cross compiler and Sconify image tool. The Data Broker has evolved into a pivotal component, orchestrating secure, policy-driven, and auditable data exchanges across distributed environments. Its advanced capabilities now include fine-grained access control, comprehensive audit logging, and seamless integration with authentication and authorization services, ensuring that strict security and compliance requirements govern all data transactions.

The security toolchain enhancements, spanning the SCONE runtime for confidential execution, the cross-compiler for enclave-ready builds, and the Sconify image tool for secure containerization, have collectively expanded confidential computing support, improved compatibility across platforms and languages, and strengthened both policy management and attestation mechanisms. This integrated approach ensures that sensitive data is protected at rest, in transit, and in use, and that only authorised, attested services can access or process data.

These improvements have been rigorously validated in real-world use cases. In the surgical scenario, the NEARDATA Data Broker enabled confidential federated learning by safeguarding sensitive laparoscopic video data, thus supporting privacy and regulatory compliance in healthcare. The Data Broker provided robust confidentiality and integrity guarantees for federated learning workflows, with performance evaluations demonstrating only modest overhead compared to non-confidential setups. In the metabolomics use case, the integration of the Confidential Compute Layer into Lithops and MinIO enabled secure, decentralized scientific computing on cloud platforms, ensuring end-to-end confidentiality from data acquisition through large-scale analysis.

Performance benchmarks and key performance indicators confirm that NEARDATA's security enhancements deliver practical and efficient protection for sensitive, distributed applications. The framework now empowers organizations to confidently develop and deploy solutions that meet the highest standards of confidentiality, integrity, and compliance, while maintaining the flexibility and performance required for modern edge-to-cloud environments. These achievements provide a solid foundation for ongoing innovation and the continued expansion of NEARDATA's security capabilities.

Use-case / Experiment	KPI	Results
Federated Learning with SCONE and Flower Framework	KPI-4	• Integrated SCONE with the Flower federated learning framework for secure, privacy-preserving model training. • Performance evaluation of a federated learning demo with SCONE showed modest overhead (10–20% increase in training time per round) compared to vanilla deployments. • Security KPI validated: confidentiality and integrity of model updates and training data are enforced, though security is hard to quantify directly. • Initial startup and response times for federated learning functions (e.g., Flower) remain within practical bounds for real-world use.
Pravega Secure I/O Operations	KPI-4	• Evaluated secure input/output operations using a sconified Pravega client in collaboration with DELL. • SCONE-protected Pravega clients show affordable latency increase (2×) at low throughput and negligible overhead at high throughput (e.g., 10k events/sec), making it practical to use in latency-sensitive applications.
Lithops with SCONE: Simplified Metabolomics	KPI-4	• Integrated the Confidential Compute Layer of the Data Broker into Lithops for spatial metabolomics annotation. • Tested on an on-premise Kubernetes cluster, confirming secure, decentralized processing of gigabytes of metabolomics data. • End-to-end confidentiality and integrity maintained throughout the workflow.
Lithops with SCONE: Keycloak Integrated	KPI-4	• Integrated Keycloak as an Identity Provider for Lithops, enabling federated authentication and policy management. • Demonstrated secure, multi-user access and policy enforcement for serverless scientific computing.

Table 14: Highlights of main KPIs achieved by the Data Broker and security tool integrations

Use-case / Experiment	KPI	Results
Federated Learning with SCONE and Flower Framework	KPI-4	• **Training Time Overhead (15%):** – Average training time per round was recorded for both the SCONE-protected and vanilla federated learning deployments. Overhead was calculated as $\frac{T_{\text{secure}} - T_{\text{baseline}}}{T_{\text{baseline}}} \times 100$, where T_{secure} and T_{baseline} are the average times per round in secure and baseline settings, respectively. • **Confidentiality (Entropy 0.98):** – Entropy of sensitive files (e.g., model updates, training data) within the TEE was measured using tools such as `ent` or `openssl`. Values close to 1 indicate strong encryption and minimal risk of data leakage. • **Availability (99.95%):** – Uptime was monitored using Prometheus or cloud dashboards over the evaluation period, and availability was calculated as $\frac{\text{Uptime}}{\text{Total Time}} \times 100$.

Table 15: Federated Learning with SCONE and Flower: Quantitative KPIs and Measurement Methods

Use-case / Experiment	KPI	Results
Lithops with SCONE: Simplified Metabolomics	KPI-4	• Native Mode: Mean execution time of 1.774 minutes. • Sconified (Simulation): Mean execution time increased to 10.046 minutes, approximately 5.662× slower compared to native mode. • Sconified (Hardware): Mean execution time further increased to 44.963 minutes, approximately 25.34× slower compared to native mode, with failed executions.

Table 16: Performance comparison for the SCONE Simplified Metabolomics use-case.

Use-case / Experiment	KPI	Results
Lithops with SCONE: Keycloak Integrated	KPI-4	1. ACVR (Access Control Violation Rate) for Access Token (AT) Requests • Total AT Requests: 200 • Failed due to Incorrect Credentials: 23 • Successful AT Issuance: 177 • Calculation: ACVR = 23 / 200 = 0.115 (or 11.5%) 2. ACVR for Validation Token (VT) Issuance After Expiration • VT Validation Attempts After Expiration: 177 • Failed (as expected, since tokens expired): 177 • Calculation: ACVR = 177 / 177 = 1 (or 100%)

Table 17: Performance comparison Lithops with SCONE Keycloak Integrated.

References

[1] E. Falcão, F. Silva, C. Pamplona, A. Melo, A. S. M. Asadujjaman, and A. Brito, "Confidential kubernetes deployment models: Architecture, security, and performance trade-offs," Applied Sciences, vol. 15, no. 18, 2025.

[2] R. Balasubramonian, J. Chang, T. Manning, J. H. Moreno, R. Murphy, R. Nair, and S. Swanson, "Near-data processing: Insights from a micro-46 workshop," IEEE Micro, vol. 34, no. 4, pp. 36–42, 2014.

[3] W. Xiong, L. Ke, D. Jankov, M. Kounavis, X. Wang, E. Northup, J. A. Yang, B. Acun, C.-J. Wu, P. T. P. Tang, G. E. Suh, X. Zhang, and H.-H. S. Lee, "Secndp: Secure near-data processing with untrusted memory," in Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security (CCS), pp. 1483–1497, ACM, 2022.

[4] N. Abbas, Y. Zhang, A. Taherkordi, and T. Skeie, "Multi-access edge computing: A review," IEEE Access, vol. 8, pp. 116974–117017, 2020.

[5] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, and J. Zhang, "Edge intelligence: Paving the last mile of artificial intelligence with edge computing," Proceedings of the IEEE, vol. 107, no. 8, pp. 1738–1762, 2019.

[6] Intel, "Intel tdx: Confidential computing for the cloud," tech. rep., Intel Corporation, 2024. Cloud deployment guide.

[7] Intel Corporation, "Intel® trust domain extensions (intel® tdx) white paper," tech. rep., Intel Corporation, 2023. Available online.

[8] AMD, "Amd sev-snp: Mitigating vm attacks," tech. rep., AMD Corporation, 2023. Security mitigation guidance.

[9] IBM Security and Ponemon Institute, "Cost of a data breach report 2023," industry report, International Business Machines Corporation, Cambridge, Massachusetts, 7 2023. Based on research conducted by the Ponemon Institute.

[10] S. Arnautov, B. Trach, F. Gregor, T. Knauth, A. Martin, C. Priebe, J. Lind, D. Muthukumaran, D. O'keeffe, M. L. Stillwell, et al., "{SCONE}: Secure linux containers with intel {SGX}," in 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16), pp. 689–703, 2016.

[11] R. Gracia-Tinedo, F. Junqueira, T. Kaitchuck, and S. Joshi, "Pravega: A tiered storage system for data streams," in Proceedings of the 24th International Middleware Conference, Middleware '23, (New York, NY, USA), p. 165–177, Association for Computing Machinery, 2023.

[12] S. Arnautov, B. Trach, F. Gregor, T. Knauth, A. Martin, C. Priebe, J. Lind, D. Muthukumaran, D. O'Keeffe, M. L. Stillwell, et al., "Scone: Secure linux containers with intel sgx," in 12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 16, pp. 689–703, 2016.

[13] F. McKeen, I. Alexandrovich, I. Anati, D. Caspi, S. Johnson, R. Leslie-Hurd, and C. Rozas, "Intel software guard extensions (intel sgx) support for dynamic memory management inside an enclave," in Proceedings of the Hardware and Architectural Support for Security and Privacy 2016, HASP '16, (New York, NY, USA), pp. 1–9, Association for Computing Machinery, 2016.

[14] "The openmessaging benchmark framework." `https://openmessaging.cloud/docs/benchmarks/`, 2018. Accessed: May 13, 2025.

[15] KIO España, "Kio españa | data center tier iv." `https://www.kionetworks.es/`, Apr. 2025. Accessed: May 13, 2025.

[16] Y. Zheng, T. Yu, et al., "bpftime: userspace ebpf runtime for uprobe, syscall and kernel-user interactions," OSDI 2025, Nov. 2023.

[17] A. P. P. Hartono, A. Brito, et al., "Crisp: Confidentiality, rollback, and integrity storage protection for confidential cloud-native computing," 2024 IEEE 17th International Conference on Cloud Computing, Aug. 2024.