HORIZON EUROPE FRAMEWORK PROGRAMME

NEARDATA

(grant agreement No 101092644)

Extreme Near-Data Processing Platform

D4.1 Data Broker release and documentation

Due date of deliverable: 30-04-2024
Actual submission date: 30-04-2024

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	49
WP/Task related to this document	WP4 / T4.2-T4.4
WP/Task responsible	TUD
Leader	André Martin (TUD)
Technical Manager	Pubudu Jayasena (TUD)
Quality Manager	André Martin (TUD)
Author(s)	André Martin (TUD), Pubudu Jayasena (TUD), Robert Krahn (TUD), André Miguel (SCO)
Partner(s) Contributing	TUD, NCT, SCO
Document ID	NEARDATA_D4.1_Public.pdf
Abstract	The deliverable describes the architecture and functionality of the Data Broker.
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	15-03-2024	André Martin (TUD), Pubudu Jayasena (TUD), Robert Krahn (TUD)	Initial draft and structure in place.
0.2	25-03-2024	André Martin (TUD), Pubudu Jayasena (TUD), André Miguel (SCO)	First Draft with federated learning/Flower Integration.
0.3	28-03-2024	Pubudu Jayasena (TUD)	Integrated measurements for network shielding and runtime performance.
0.4	15-04-2024	Pubudu Jayasena (TUD)	Rewrote introduction
0.5	20-04-2024	André Miguel (SCO)	Incorporated KeyCloak and Lithops integration.
1.0	30-04-2024	André Martin (TUD), Pubudu Jayasena (TUD), André Miguel (SCO)	Final pass and release.

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

This document presents the initial release and documentation of the Data Broker, a key component of NEARDATA that facilitates the orchestration of trustworthy data flows across the Cloud/Edge continuum. An important goal of the Data Broker is to provide secure data governance, including data access and confidential data transfer, for distributed Data Sources in the Data Catalog. The Data Broker component was designed to enable stakeholders to securely access and share data as well as to ensure confidentiality and integrity for applications and their components. The Data Broker provides furthermore mechanisms to manage user access policies and roles, and several mechanisms for secretly accessing data.

The primary goal of this document is to provide a detailed overview of the components that make up the Data Broker as a whole, and how security is being enforced using policies. Furthermore, the document describes how the Data Broker is integrated into existing frameworks and tools used in NEARDATA. Finally, a preliminary evaluation was conducted to validate various design decisions and the performance of the system.

In order to showcase how the Data Broker controls data access, we provide a first prototypical implementation and integration of a federated learning application using FLOWER, an open-source framework, and SCONE, the framework we developed to run applications in a trustworthy manner using Trusted Execution Environments (TEEs). The example is referenced throughout the whole document to highlight the various security aspects and mechanisms developed within NEARDATA, showcasing how they provide confidentiality and integrity for this specific scenario. Although federated learning provides already some degree of confidentiality through local learning and sharing only model parameters, additional mechanisms are necessary to establish full trust as required and set out by the NEARDATA project.

After the description of the federated learning example, we provide a thorough analysis of the threat model and potential attacks we envision in NEARDATA. Based on these findings, we then present the so-called policy definitions that allow application developers and users to define their access policies and how data should be protected. In addition to the established policies, we furthermore outline protection goals derived from prior research on threats and vulnerabilities commonly encountered in Artificial Intelligence (AI) systems operating in these settings.

Besides the policy definition, we will then present the various security tools we developed in order to control data access and provide confidentiality and integrity for the applications running within NEARDATA. The security tools range from SCONE runtime, which enables the application to run in TEEs, a cross-compiler, and the Sconify tool, both enabling developers to convert native apps into confidential ones. The deliverable concludes with a summary of achievements.

2 Introduction

The NEARDATA "Extreme Near-Data Processing Platform" aims to establish a sophisticated infrastructure that facilitates data transfer across Object Storage and Data Analytics systems along the Compute Continuum. In this deliverable, we provide an in-depth overview of Data Broker including its primary sub-components that execute the system's functions and how it integrates with other components of the NEARDATA architecture.

NEARDATA's design has been derived from the Reference design of International Data Spaces [1], which has served as a significant source of inspiration. The recommended architecture, which is based on the actual components given by each of the NEARDATA partners (refer to D2.2 NEARDATA Architecture Specs and Early Prototypes). The advanced near-data processing platform comprises four modules, as described in D2.2 NEARDATA Architecture Specs and Early Prototypes. These modules, namely Data Plane / XtremeHub (refer to D3.1 XtremeHub first release and documentation), Control Plane, Analytics, and Data Sources, are designed to cater to the specific requirements and functions of NEARDATA components. The standard architecture has six fundamental entities that are seamlessly incorporated into a collection of modules: Data Providers, Data Consumers, Data Connectors, Data Broker, App Store, and Identity Provider.

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. An integrated view of Data Broker is depicted in Figure 1

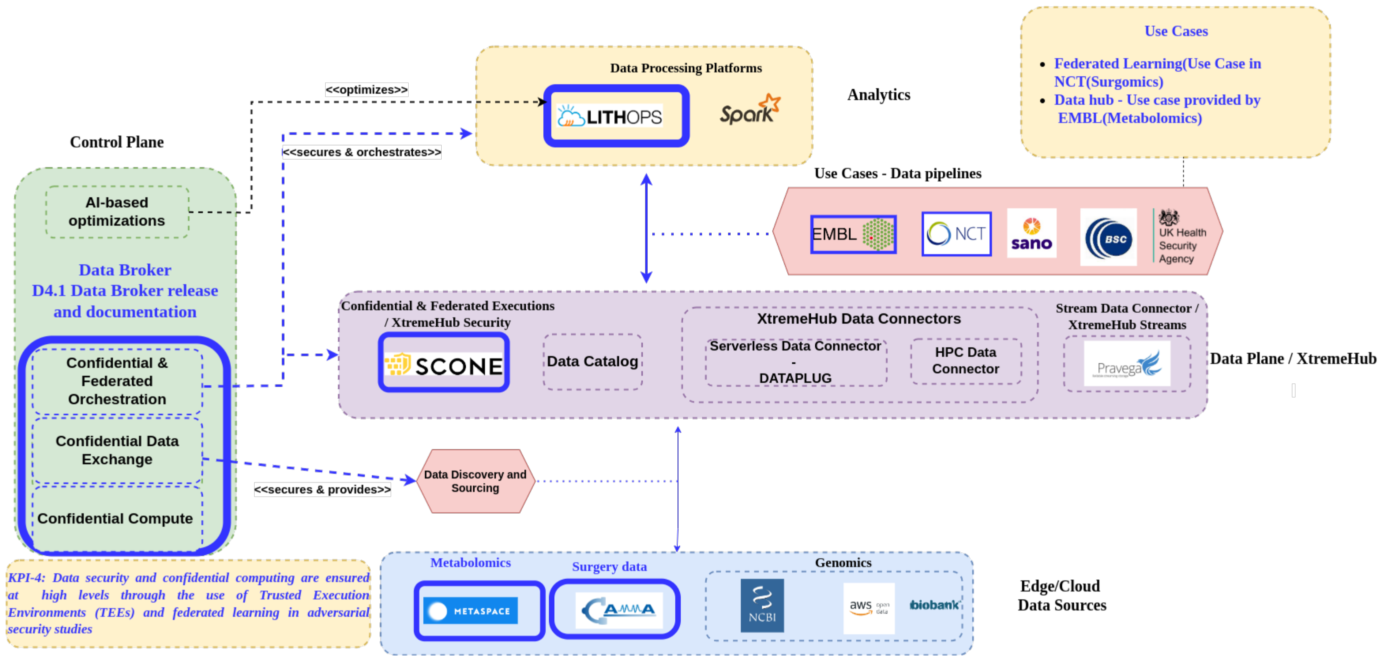

Use Cases
Federated Learning(Use Case in NCT(Surgomics)
Data hub - Use case provided by EMBL(Metabolomics)
Data Processing Platforms
LITHOPS
Spark
Analytics
<<optimizes>>
<<secures & orchestrates>>
Control Plane
AI-based optimizations
Data Broker
D4.1 Data Broker release and documentation
Confidential & Federated Orchestration
Confidential Data Exchange
Confidential Compute
Use Cases - Data pipelines
EMBL
NCT
sano
BSC
UK Health Security Agency
Confidential & Federated Executions / XtremeHub Security
SCONE
Data Catalog
XtremeHub Data Connectors
Serverless Data Connector - DATAPLUG
HPC Data Connector
Stream Data Connector / XtremeHub Streams
Pravega
Data Plane / XtremeHub
<<secures & provides>>
Data Discovery and Sourcing
KPI-4: Data security and confidential computing are ensured at high levels through the use of Trusted Execution Environments (TEEs) and federated learning in adversarial security studies
Metabolomics
METASPACE
Surgery data
CAMMA
Genomics
NCBI
aws open data
biobank
Edge/Cloud Data Sources

Figure 1: High-level overview of Data Broker

NEARDATA aims to tackle the conventional issue associated with data analysis by expanding the notion of DataHub to handle large-scale data. The innovative XtremeHub will have dependable and high-performing data connections, enabling data analytics systems to access and explore extreme data stored in Object Storage. The platform will have the capability to effectively coordinate various workflows and data analysis platforms using optimised AI algorithms. Additionally, in order to meet the specific requirements of handling sensitive health data, the Data plane/Xtreame hub con-

[1] `https://internationaldataspaces.org/`

nected with Data Broker will provide secure and privacy-preserving solutions by utilising TEE-based technology to deploy protected data flows across the entire Compute Continuum. The Data Broker service restricts access to the data items and streams, providing safe access to data using TEEs. The NEARDATA platform is a cutting-edge technology specifically developed for extracting valuable insights from large and dispersed unstructured data collections. It may be used in both cloud and edge computing environments, such as High-Performance Computing (HPC) and Internet of Things (IoT) devices. It incorporates advanced AI technology and offers a distinct confidential cybersecurity layer to ensure safe data processing.

This document includes four main parts:

1. First section provides an overview of the use cases that are improved through the integration of a Data Broker, as well as a detailed explanation of a federated example application. This example which will be used throughout the document as a running example in order to demonstrate the various security-related features we are developing within NEARDATA.

2. The NEARDATA Security Overview provides a summary about potential threats and attacks in the context of Streaming and AI applications as well as presents a policy definition which consists of protection goals as well as protection mechanisms in order to address the previously analyzed threats.

3. The First Release of the Data Broker section provides details about the technological components involved and how they have been evolved in order to support the security requirements. The section is subdivided into runtime security as well as network security addressing different layers of the NEARDATA architecture.

4. In the Conclusion section, we summarize our contributions and outline the ongoing and future work.

3 Use Cases Enhanced through Data Broker Integration

3.1 Example Application: Federated Learning - NCT Use Case (Surgomics)

In this section, we will briefly describe an example application that we will use throughout this document to describe the security tools we have implemented within the scope of WP4 for NEARDATA. As an example, we chose federated learning as it is used by one of the use case providers NCT for Computer-assisted surgery, and it demonstrates nicely how multiple parties collaboratively work together without having to trust each other as well as using infrastructures such as cloud environments and edge devices which are under a third-party administrative domain that cannot be trusted. Furthermore, it utilizes AI techniques and therefore it is well suited as a demonstrator in the context of NEARDATA.

Federated learning [1] is an emerging machine learning technique which allows participating clients to collaboratively train a joint global machine learning model without sharing their local training data. Federated learning reduces privacy risks for the local training data which may be highly sensitive relating to personal finances, political views, health, etc. Thus, it has been widely used in the industry since it helps companies to comply with regulations on issues regarding how personal data is handled and processed such as the EU's General Data Protection Regulation (GDPR) [2].

The core idea of federated learning is that each client trains a local model, rather than sharing training data to a centralized training system which is deployed in an untrusted environment, e.g., a public cloud. For each iteration, the clients send their local model training parameters (i.e., local computed gradients) to the central system to train a global model which takes benefits from all local training data from clients. Typically, the central system aggregates the local training gradients from the clients and sends the aggregated gradients back to them. This training process is repeated until it converges or the global model reaches a certain desired accuracy.

An example of federated learning in real-life deployment is that several hospitals collaborate to develop a shared machine-learning model based on their patient data to detect a disease at an early stage. Each hospital processes its data locally, shares the local gradients with the central training system, and receives the global gradients in each iteration.

While promising at first glance, the federated learning paradigm suffers from several vulnerabilities:

1. An attacker with privileged/root accesses can easily obtain the training models. The attacker can also compromise the privacy of individuals in the training data by inferring it from parameters of the global model [3]. Therefore, the training models need to be protected at **rest**, in **transit**, and in **use**.

2. A large number of malicious clients may collude with each other to reveal local data and local models of the remaining clients [4].

3. Malicious clients can tamper with their local training data or parameter updates forwarded to the central training system to corrupt the global model[5, 6].

4. The attacker compromises the sensor on the field and provides tampered data to the client.

To handle issues (1) and (2), state-of-the-art solutions rely on a privacy-preserving mechanism such as differential privacy or secure multiparty computation. The disadvantage of the differential privacy mechanism is that it reduces the performance of the global training model regarding utility or accuracy. Meanwhile, the solutions based on secure multiparty computation incur significant overhead [7, 8]. To cope with issue (3), several Byzantine-robust federated learning mechanisms have been proposed [9, 10, 11, 6]. The core idea behind these mechanisms is to reduce the impact of statistical outliers during model updates in the federated learning system. However, recent works [9, 11, 6] show that the mitigation of the impact is still not enough to protect the utility of the global model. The malicious clients can still affect the accuracy of the global model trained by a Byzantine

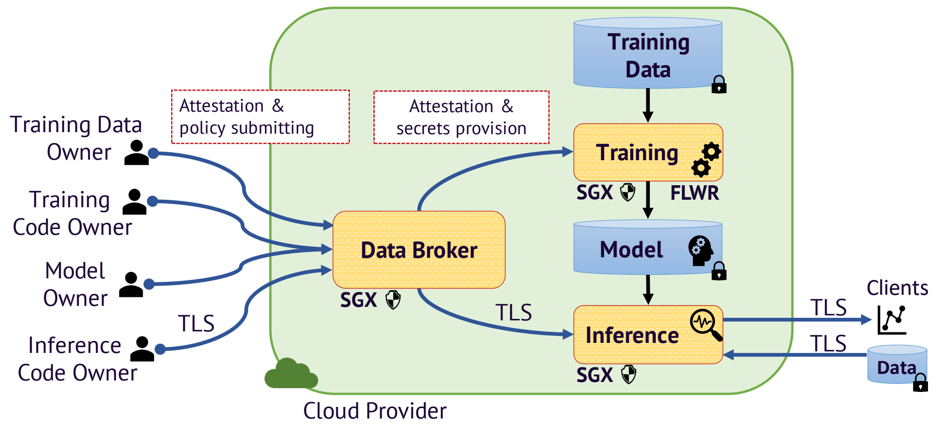


Figure 2: Federated Learning Architecture 1 in NEARDATA.

robust mechanism by carefully tampering with their model parameters sent to the central training system [12].

In NEARDATA, we overcome these limitations by building a confidential federated learning system using TEEs, e.g., Intel SGX. Trusted Execution Environment (TEE) technologies, such as Intel Software Guard eXtensions (SGX) have gained much attention in the industry [13, 14] as well as in academia [15, 16, 9, 17, 8, 18, 19, 20, 21, 22, 23]. To ensure the confidentiality and integrity of applications, TEEs execute their code and data inside an encrypted memory region called an enclave. Adversaries with privileged access cannot read or interfere with the memory region and only the processor can decrypt and execute the application inside an enclave. In addition, TEEs such as Intel SGX also provide a mechanism for users to verify that the TEE is genuine and that an adversary did not alter their application running inside TEE enclaves. The verification process is called Remote Attestation [24] and allows users to establish trust in their application running inside an enclave on a remote host.

We leverage TEEs to handle issue (1), by providing end-to-end encryption. Also, our solution encrypts input training data and code (e.g., Python code) and performs all training computations including local training and global training inside TEE enclaves. The secure federated learning enables all gradients updates via Transport Layer Security (TLS) connections between the enclave of clients and the enclaves of the central training computation. Thus attackers with privileged accesses cannot violate the integrity and confidentiality of the input training data, code, and models. We also ensure the freshness of the input training data, models, and, by applying an advanced asynchronous monotonic counter service [25].

We tackle issues (2) and (3) by developing a Data Broker component integrated with Configuration and Attestation Service(CAS) based on the remote attestation mechanism supported by TEEs [25, 26]. The component ensures the integrity of input data and training code, i.e., it makes sure that training computations are running with correct code, correct input data and not modified by anyone, e.g., an attacker or malicious client. This component also monitors and attests to the compliance of participated clients with the pre-defined agreement before collaborating to train the global machine learning model. In addition, it can clone the global training computation and randomly take a sample of clients for the training computation. This helps to detect outliers regarding the utility which helps to solve issue (3). Our preliminary evaluation shows that we can ensure the confidentiality and integrity of federated learning computations while maintaining the same utility/accuracy of the training computations.

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 neces-

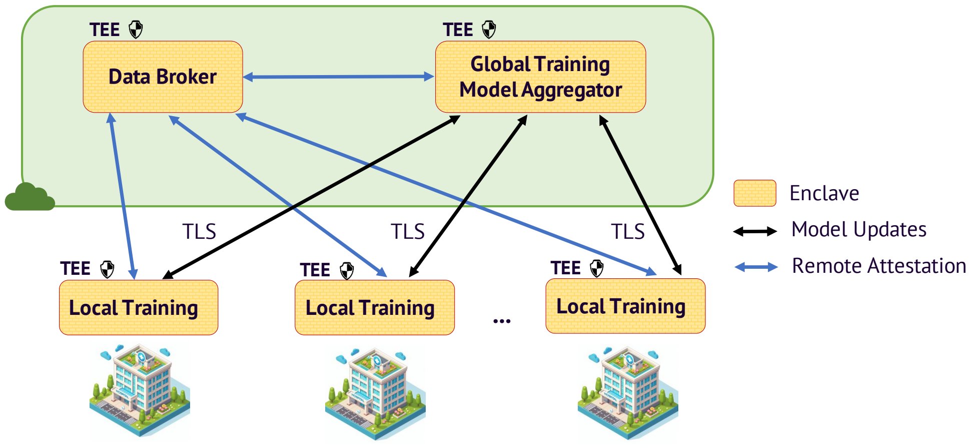


Figure 3: Federated Learning Architecture 2 in NEARDATA.

sarily 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. Examples of the policies can be found in [26].

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 [27] - 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 [13, 28].

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.***(refer D2.2 NEARDATA Architecture Specs and Early Prototypes)***

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 accomplishments are assessed based on the Key Performance Indicators (KPIs) provided in the Description of the Action (DoA).

KPI-4: Data security and confidential computing are ensured at high levels through the use of Trusted Execution Environments (TEEs) and federated learning in adversarial security studies.

Use-Case	KPI	Result
Surgery	KPI-4	We achieved this Key Performance Indicator (KPI) by following the approach outlined below. ***(refer D5.1 First release of KPI benchmarks in all use cases and data connector libraries)*** • Examine how confidential federated learning applications, which utilise Trusted Execution Environments (TEEs), tackle the inherent difficulties in the federated learning paradigm. • Assess the effectiveness of the confidential federated learning application by analysing the impact of SCONE's network shield, configured through the Data Broker's CAS component, on execution time. Compare this impact to both the Vanilla (Intel/Non-SGX) environment and the SCONE (Intel SGX/hardware mode) setup, with a particular focus on the execution time of federated learning. • Assess the impact of optimisations on the initial response time of functions in the FL application framework.

Table 1: Highlights of main KPIs achieved by federated learning use-case

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 2: Use-cases source codes released.

3.2 Example Application: Data hub - Use Case provided by EMBL(Metabolomics)

The NEARDATA Metabolomics Data Space aims to become a worldwide reference in the field of metabolomics. We acknowledge the significant efforts made by EMBL, a pioneer in this field, to use the METASPACE platform as the primary means of analysing, disseminating, and visualising spatial metabolomics data.

Frequently, researchers in the field of omics are interested in handling Confidential data. In order to adhere to the privacy requirements of customised annotation tasks, our objective is to execute the METASPACE annotation code within secure enclaves rather than stateless serverless components. Security will be provided using SCONE containers, which will be scheduled using Lithops on a Kubernetes (K8S) backend.

The Data Broker will be in charge of orchestrating the TEEs deployed with SCONE to protect the executions with Lithops on METASPAC. Furthermore, the utilisation of AI-based optimisations will assist us in establishing an effective workflow on our data processing platform.

Deploying a machine learning pipeline architecture inside a secure and trustworthy environment. Privacy and security will be ensured by employing SCONE containers on a secret K8S backend, together with a Lithops orchestrator. The majority of OMICs data is subject to stringent confidentiality standards that are not compatible with public cloud services. Utilising on-premise cluster resources to execute the pipeline at the edge will effectively cater to the requirements of private workloads.

The current level of this use case is preliminary, but, we have effectively created an initial prototype that incorporates Lithops, METASPACE, and Data Broker containers. We employed the KIO testbed to accomplish this goal, taking advantage of its built-in K8S deployment to develop a functional architecture for the final version of the use-case. The final proposal will include the confidential version of the pipeline to manage the classified datasets from METASPACE and transmit the results to the METASPACE database and the ElasticSearch engine.

Use-Case	**KPI**	**Result**
Metabolomics	KPI-4	We have integrated Data Broker Confidential Compute Layer component(SCONE) into Lithops and tested its early usage on an on-premise K8s cluster

Table 3: Highlights of main KPIs achieved by Data-hub use-case

4 NEARDATA DataBroker Overview

The Data Broker provides user authentication through mechanisms such as mutual TLS handshakes as well as an attestation that grants access to confidential data, which can be decrypted and manipulated. Additionally, the Data Broker includes orchestration services and connects different types of data streams of analytical platforms and services. The Data Broker comprises three primary components:

1. **Confidential Compute Layer:** The main function of this task is to develop mechanisms to process high volumes of data within trusted execution environments such that confidentiality, integrity and freshness of the data can be guaranteed.

2. **Confidential Data Exchange:** The main function of this task is to develop mechanisms to exchange data through the network in a confidential manner.

3. **Confidential and Federated Learning orchestration layer:** The main function of the Confidential orchestration layer is to develop mechanisms to orchestrate various diverse applications securely and confidentially. It will allow the use of state-of-the-art orchestration tools such as Kubernetes however provide trust such that deployed applications and processes on the infrastructure cannot be tampered with. The federated learning orchestration layer provides a structure for coordinating the processing of data in situations including federated learning. The primary premise is that the data used for machine learning training remains at its source (divided among several clients). The local model is trained independently at each client, and then the model is transmitted to the central server for aggregation and averaging. This methodology ensures the secure handling of medical data, preventing it from being transferred across institutional or national boundaries. The framework will utilise the existing infrastructure established inside the project, encompassing the data storage, processing capabilities, and interfaces.

The following discussion will focus on the threat model we consider for NEARDATA which we will then use to derive requirements from as well as to define security policies which are applied for the different stakeholders in NEARDATA. Besides the security policies, which are high-level definitions and requirements originating from the users, we then introduce the session language used to provide more fine-graned configuration options. After the introduction of the policies, the architecture of the different components working with those policies is being presented.

4.1 Threat Model

In NEARDATA we target to harness cloud resources as well as edge clouds and devices to conduct data analytic tasks in addition to traditional cloud resources, which shifts a significant part of the workload from the Cloud/Edge devices. Due to the rapid growth in streaming data volume, many applications are leveraging edge cloud platforms these days to store and analyze data efficiently. Such platforms must be trusted to protect data privacy and security from malicious insiders or outside attacks when hosting applications in these environments.

For the threat model, we consider classical cloud applications as well as edge platform services capturing and analyzing, e.g. telemetry data. We also recognize the significance of mission-critical IoT with tight control loops but do not target them in our threat model. Our target scenario includes source sensors, edge platforms, and cloud servers. We assume a potentially malicious environment in which privileged processes such as the operating system have full control over system call arguments and their results. In such a compromised system, an attacker can not only modify the system data but can also eavesdrop on system activities. Apart from that, we assume that access to the hardware is strictly regulated and that an adversary cannot mount physical attacks on the otherwise trusted CPU.

We consider the scenarios of malicious field nodes trying to access other users' resources and malicious remote nodes trying to access data from field IoT devices in addition to regular cloud nodes.

The first scenario includes legitimate, but compromised, nodes trying to access resources and applications in the edge node or in the cloud for which they are not authorized. In the second scenario, an external attacker tries to access data generated by devices in the field or even take control of the devices. Also in this scenario, the attacker can be authorized to access some data but wants to access data or perform actions for which they have no authorization. We furthermore assume untrusted edge-cloud links that require encryption of uploaded data. We consider malicious adversaries capable of identifying IoT data, tampering with edge processing outcomes, or obstructing processing progress as in-scope threats. Based on the assumption that powerful adversaries exist: they control all applications on the edge by exploiting weak configurations or bugs in the edge software; they control the entire OS as well as all applications on the edge. For cloud nodes, we assume an adversary has full control over all processes and all hardware components except the CPU, based on the Intel's SGX threat assumption. The adversary can check the memory trace of all processes except those running within the enclaves. Also, they can monitor or interfere with the communication between edge servers and cloud servers.

In addition, the system operates under a specific threat model that assumes specified conditions. The software is created and disseminated by a trusted developer, and a reliable signer is responsible for preparing the software for execution within SGX. This process involves utilizing a trustworthy SGX framework that facilitates the execution of legacy software within SGX, with the signer generating a valid SigStruct. Users then customize the SGX-enabled software through secure means provided by the SGX framework, encompassing configuration elements such as files, environment variables, and program arguments. It is crucial to note that the developer, signer, and user, along with their respective systems, are assumed to be benign and possess secure channels for exchanging information regarding the software's integrity. The customized software is subsequently transmitted and executed within an SGX-enabled environment, even in the presence of potential adversaries. The security goals of the SGX environment, including the trustworthiness of the remote attestation mechanism and infrastructure, are integral to this model. Additionally, we operate under the assumption that the SGX processor implementation is not vulnerable to exploitation, and the adversary is knowledgeable about the attestation protocols, eliminating security through obstruction.

4.2 Attacks in The Context of NEARDATA

Considering the above-listed threats, a malicious user can drive the following attacks in the context of NEARDATA applications. To gain access to sensitive data that is used for training purposes such as radiomics imaging data, etc., an infrastructure administrator with root privileges can simply copy the locally stored files out of the running VM image. Although this attack can be prevented by using end-to-end encryption, i.e., encrypting the files beforehand, the Python processes running the training software such as Pytorch, TensorFlow, etc., need to access this data, i.e., require access to the private key used to encrypt the data at rest. An attacker can therefore create a memory dump of these processes in order to reveal the key and perform the en/decryption him-/herself.

It is also worth noting that training data is a precious resource that can be monetized and conveyed to multiple entities also in real-time directly from the edge. As a variation of the above attack, an external malicious user can attempt to access data from the field devices by exploiting their low complexity and lack of support for fine-grained access control policies. In NEARDATA, we tackle this attack by means of our Data Broker component, which intermediates any exchange between the devices and the users.

Besides gaining access to confidential data, another attack vector is the malicious introduction of wrong information in order to tamper with training results. This can be achieved by a malicious user pretending to be a legitimate collaborator if we consider federated learning. Although this attack seems to be not that easy at first as it requires access to certificates as well as keys to pass the mutual authentication when establishing Transmission Control Protocol(TCP) connections between the collaborator as well as the aggregator, such keys, as well as certificates, can be easily retrieved as described previously.

Another way of tampering with training results is through the modification of the training code itself. This type of attack can be prevented through the use of integrity protection mechanisms at the file system level such that the code is signed beforehand. Another type of attack are so-called rollback attacks: For these attacks, a malicious user provides the software with an older version of either the training data or the trained model such that, e.g., classifiers do not correctly detect/recognize certain items any more. This requires, as before, access to the file system as well as the capability to stop processes and resume them which is easily achievable by administrators with root privileges and hypervisor access.

In stream processing systems, network communication patterns directly reflect the structure of the streaming applications. These applications typically consist of multiple processing stages organized into a Directed Acyclic Graph (DAG) that runs on a collection of networked machines. In general, each stage is partitioned into multiple nodes that are executed in parallel. Each node performs local computations on the input streams from its in-edges and produces output streams to its out-edges. By observing network-level communication between the different stages of the DAG, an adversary may be able to extract information about the data being processed by the application. In addition, stream processing systems are susceptible to side-channel attacks, which compromise users' data security and privacy using any publicly accessible information that is not privacy-sensitive, namely side-channel information. Such public information is typically correlated "secretly" with certain privacy-sensitive data that should be protected. Attackers then explore the hidden correlations to finally infer the protected data from the side channels. Since any public information can have the potential to link to some sensitive data, side-channel attacks can happen anywhere in the edge computing architecture. These side-channel attacks happen both at the memory level and network level.

4.3 Security Policies

Security policies in NEARDATA are used as input for the Data Broker component and define what level of protection the application user and developer desire for their application in order to mitigate the previously described attacks. The policies are furthermore used to control access to data items and restrict them only to trusted parties. To define those policies, we will now list the several attacks and what mechanism mitigates them:

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 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** & **integrity protection** for data in transit)	Data Broker-Confidential Data Exchange Layer Network Protection Shield (NPS)
. Reading & modifying or forging input data for training aswell as training models (**confidentiality protection** & **integrity protection**, **lateral movement, tampering with training results** for data at rest)	Mobile device authentication

Table 4: Attacks and security mechanisms

4.4 Policy Definitions

In NEARDATA, application developers and users can describe their constraints with regard to security policies in two ways: Either they choose the attacks they want their application to be protected against as called protection goals, or they choose the concrete mechanisms that should be enabled while the application is running.

We will now lists the different protections goals that can be chosen by application users or developers and their mappings to the respective mechanism provided by the security tools of NEARDATA:

Protection	Mechanisms
Confidentiality for data being processed **confProc**	Trusted Execution Environment (TEE) or, if not available, secure and measured boot
Integrity for data being processed **integrityProc**	Trusted Execution Environment (TEE) or, if not available, secure and measured boot
Confidentiality for data at rest **confRest**	File system protection shield (encrypted volumes)
Integrity for data at rest **integrityRest**	File system protection shield (integrity protected volumes)
Confidentiality for data at transit **confTrans**	Network protection shield (NPS)
Integrity for data at transit **integrityTrans**	Network protection shield (NPS)

Table 5: Protection Goals / Options

In case the nodes do not provide TEE support such as when using accelerators like GPUs, secure and measured boot can be used in order to provide a minimum of security. Secure boot ensures that the BIOS configuration is current with regards to the devices the operating system, etc. is booted from while measure boot ensures that the operating system is booted correctly by exhibiting signatures for each step during the booting processes that can be attested afterwards.

In the case the application user or developer is familiar with the mechanism, he or she can also directly specify which protection mechanisms should be enabled instead. It is also possible to specify both, i.e., a protection mechanism as well as a protection goal. The system will then create an intersection between the chosen protection goals and their mappings as well as the explicitly specified protection mechanisms. The following definition is an example where an application developer used protection goals as well as protection mechanisms to specify his needs from the security tools:

```
confRest: true
integrityTrans: true
tee: true
```

As shown in the example above, the user chose as a protection goal to provide confidentiality for the data at rest, integrity protection for data exchanged between processes and services Furthermore, he opted that the processes should run within a trusted execution environment. Based on the above definitions and protection goals, the application will utilize the file system shield to provide confidentiality for all files created, read and written by the application as well as network shield to cover the integrity protection goal for data at transit. Furthermore, the processes will also run a TEE as explicitly specified in the description providing confidentiality as well as integrity protection for data being processed although this was not explicitly specified as a protection goal by the user.

4.5 Fine Grained Policy Definitions

The previously defined policy definitions can be considered as high-level definitions which does not allow the interaction with other processes and components and limit the data access solely to its own entities. Hence, they describe the protection goals for an entire application which comprises a set of processes and services. However, it is often necessary to perform more fine-grained adjustments in terms of allowing other processes or components to access shared data, or as in the federated learning use case, where we do not want to have encryption and integrity protection for all directories for sharing or performance reasons. Another reason is that several protection mechanisms cannot be automatically inferred. For example, two services of an application should perform mutual authentication using TLS. In order to make this work, it is necessary to define what port a certain service is listening on and from what other service that service is granting connections/accesses to. Take as an example the federated learning use case. In federated learning, a single server process performs communication with a set of clients processes. In order to prevent tampering with the gradients exchanged during the learning phase, only authenticated client processes are allowed to join the federation, i.e., only after a successful TLS handshake as well as authentication by presenting the proper certificates. This requires a fine-grained configuration of the different entities, i.e., client processes as well as the server processes. We therefore merge in our Data Broker component high-level policy definitions with fine-grained ones in order to establish the correct properties. We will now present the policy definition that serves as input of the Data Broker component of NEARDATA. The policy definition is carried out as a so-called session language used to create session descriptions that entail all security-relevant details of an application. An application in NEARDATA that wants to exchange data with other entities can have polices that consist one or several session descriptions. Each session description defines a set of

- **services** that are part of the application,
- **secrets** that are securely stored and passed to the services,
- **images** that define which regions of a container (image) are encrypted or authenticated,
- **volumes** which are like Docker volumes but encrypted, and
- **access policy** that defines who can read or modify the session description.

The session language uses a YAML-like syntax and is similar to the syntax used in docker-compose files as shown below. In the following, we will use the term session description and session policy (or just policy) interchangeably.

```

name: NEARDATA_DEMO_46-16503
version: "0.3.9"

security:
 attestation:
    mode: none
    tolerate: [debug-mode, hyperthreading, outdated-tcb, software-hardening-needed]
    ignore_advisories: "*"

services:
    - name: python_server
    mrenclaves: [825b4b54233d060a408edda8ac761f1efa4b504a30b89d02b83ab7fb07baeb88

volumes:
    - name: my_database1

images:
    - name: prep_image
```

```
volumes:
    - name: my_database1
    path: /app/input

secrets:
    - name: ca_private_key
      kind: private-key
    - name: ca_certificate
      kind: x509-ca
      private_key: ca_private_key
      common_name: Flower_CA

    - name: server_private_key
      kind: private-key
    - name: server_certificate
      kind: x509
      private_key: server_private_key
      issuer: ca_certificate
      common_name: flower_demo_server
      endpoint: server
      dns: localhost

  - name: client_private_key
    kind: private-key
  - name: client_certificate
    kind: x509
    private_key: client_private_key
    issuer: ca_certificate
    common_name: flower_demo_client
 endpoint: client

```

The above description shows a policy example to configure the federated learning prototype so that certificates and keys are generated and injected in order to perform mutual authentication using TLS. Furthermore, it defines the process that should be executed as well as runtime parameters such as the working directory. The generated secrets are defined in the secrets section of the policy file while in the injection_files section of this configuration, the paths to the different files for the injected secrets are defined. Moreover, the example shows how one creates a certificate for a certificate authority (*ca_private_key*) and how it is used to sign certificates created for one client (*client_private_key)* and one server (*server_ private_key*). In addition to the certificates, private keys are made accessible to processes running with the SCONE runtime such that the Flower-based processes can use them to perform a mutual authentication themselves.

4.6 Data Broker Components and Architecture

Figure 4 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.

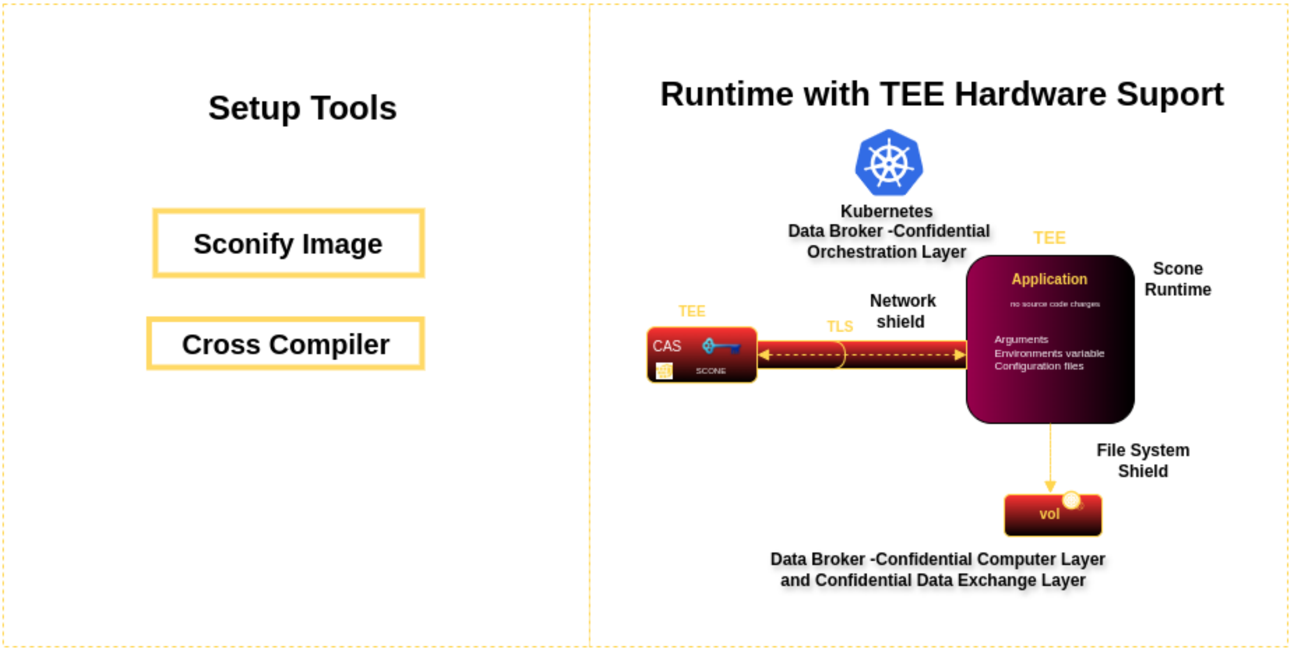


Figure 4: Security tools architecture.

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 Attestion 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:

- Watches for new deployments and monitors the Kubernetes event bus to detect new deployments.
- **Matches Labels and Annotations:** Checks for matching labels and annotations to determine if an application is covered by the specified policy definitions.
- **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.

4.6.1 Metabolomics Integration using Lithops

In the following section, we will briefly describe our technical advancements and activities for the integration of Lithops used in the Metabolomics use case. The objective is to process the data securely using Lithops. In order to do so, we need to run Lithops in Trusted Execution Environments such as Intel SGX.

With two backend execution modes (*localhost* and *serverless*), Lithops can run in TEEs through SCONE, our approach for confidential computate.

We will now describe the efforts undertaken to run such applications in a TEE:

A program that is based on Lithops is typically started up from any client workstation; if its configurations points to `localhost` backend, it is executed locally, if it is pointed to `k8s`, i.e. Kubernetes, is is executed remotely as serverless.

In the Kubernetes-based configuration, Lithops will dispatch a Job with a slice of processing to be performed on the cluster determined in its configurations. Once reaching there, Kubernetes will create the jobs and containers with the specifed Docker image, and allocate resources accordingly to perform the task. At this point, the execution turns to "localhost", hence benefiting from the TEE available to the cluster. When finished, the result goes back the client.

Two Docker images were prepared to compare the non-confidential execution (Vanilla Lithops) with the confidential execution (SCONE Lithops) as a Proof Of Concept (POC). The objective was to analyse the execution time and to spot where bottlenecks arise in confidential execution, adjust configurations, and to assert whether the execution can be successfully be inspected with privileged access from within the cluster. SCONE requires a significant larger amount of memory to work. Confidential computing is characterisitc for its unavoidable tradeoff between performance and confidentiality. A Vanilla Lithops container can work with low setting, such as *20% CPU* and *128MB RAM*, and limit itself at this. On the other hand, the SCONE Lithops container had to be modified to start with *20% CPU* and *4GB RAM*, up to the limit of **8 CPUs** and **8GB RAM**. An important environment variable that had to be added is `SCONE_HEAP=768M`, it represents the initial amount of heap memory to be loaded at the enclave creation time. The system may use more than that during its execution and all is protected with the help of TEE.

Several other modifications were required to reassure if a dispatched execution would benefit from SCONE and the TEE properly. SCONE Lithops has to learn it is supposed to use TEE. This is done by setting the environment variable `SCONE_MODE=HW`, along with the resource `sgx.k8s.io/sgx: "1"`, that will tell Kubernetes to provision the device `/dev/sgx_enclave` to the container. And note that `SCONE_MODE=AUTO` is also acceptable.

The POC has three execution configurations, differentiated by backend (localhost and Kubernetes and synchronization storage (localhost, minIO and Redis): *1) localhost*, with localhost and localhost; *2) minIO*, with Kubernetes and minIO; and *3) Redis*, with Kubernetes and Redis. A benchmark with 28 selected programs from Lithops repository[2], consisting of functions and code constructs, such as `map()`, `reduce()`, objects `map_reduce()`, `call_async()`, chained `map().map().map()` etc. was carried out for the three execution configurations. The client side does not require the usage of TEE; on the cluster side, the SCONE containers will benefit of the use of TEEs while the Vanilla containers will not use them at all. The storage system minIO is not covered by SCONE, however, Redis is covered by SCONE and running with TEE support. For this POC, no TLS network communication was employed since the objective is to adjust configurations for Lithops containers as mentioned above. See diagram 5.

The mean execution times are displayed in Table. 6. In localhost backend mode the synchronization storage is the local disk itself and it is an onerous activity. Using minIO it can be seen that, for Vanilla, it stands atop as the longest duration. Redis is the fastest of all, primarily because of its quick responses due to its in-memory database. Once the execution is started in the cluster, either Vanilla or SCONE, synchronization storage remains as expected; it is key for good performance.

Confidentiality is maintained in Lithops container, regardless of the backend being localhost or

[2]https://github.com/lithops-cloud/lithops/tree/master/examples

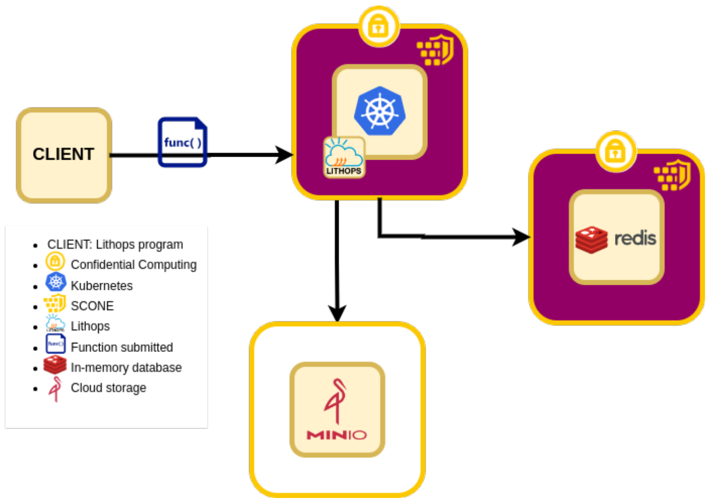


Figure 5: SCONE Lithops execution.

serverless (*please refer to D3.1 XtremeHub first release and documentation, section 6.2 Lithops Experimental Execution*). Attestation in serverless mode is under development. Lithops relies on a dynamically set environment variable `MASTER_POD_IP` to be present in container at startup. When attested, any application will ignore any environment variable, with the exception of a few `SCONE_*` that are required to know where to look for the attestation policies and calculate the enclave measurement. All the needed environment variables are provisioned by policies upon successfully attested against CAS. Therefore, to enable attestation, deeper changes are required in Lithops.

Mode	Vanilla	SCONE
localhost	5.899675	82.570785
minIO	8.205250	44.253588
Redis	4.830252	42.257455

Table 6: Mean execution times (in seconds).

4.6.2 Confidential Identity and Access Manager

A variety of applications in NEARDATA's stack can be covered by the protection of SCONE-native mechanisms. Some of them have standalone operations, i.e. attested services processing requests sent by other attested applications provisioned by the same set of policies configured in CAS. However, to cover applications with different users with different clearances to access data, which in turn require different access levels to sources of the information they handle, an auxiliary system can be employed: *Keycloak*.

Keycloak is an Identity and Access Manager – IAM that is used by applications to outsource user management. It provides the setting up of roles a user can have, according to their clearance level given by their superiors or their role in a particular process. For example, let's say that user *Alice* can submit one Lithops program to process in a confidential *Kubernetes* cluster; afterwards she will use *Data Broker* to forward those results to *Bob* – and this is exactly what Alice has clearance to do. An alternate scenario is Alice's access has been compromised by an adversary impersonating her; this attacker will do the first step of submitting program to Kubernetes unsuspiciously, but instead of forwarding results to Bob, the attacker will ship it to another recipient, *Charlie* instead, in hopes to corrupt Charlie's processes somehow or, perhaps worse, propagate spurious data that without other means of verification will be considered valid [6]. So in this case, despite the communication between entities are not permitted or even forbidden, without previous verification it goes unchecked to the destiny intended by the attacker.

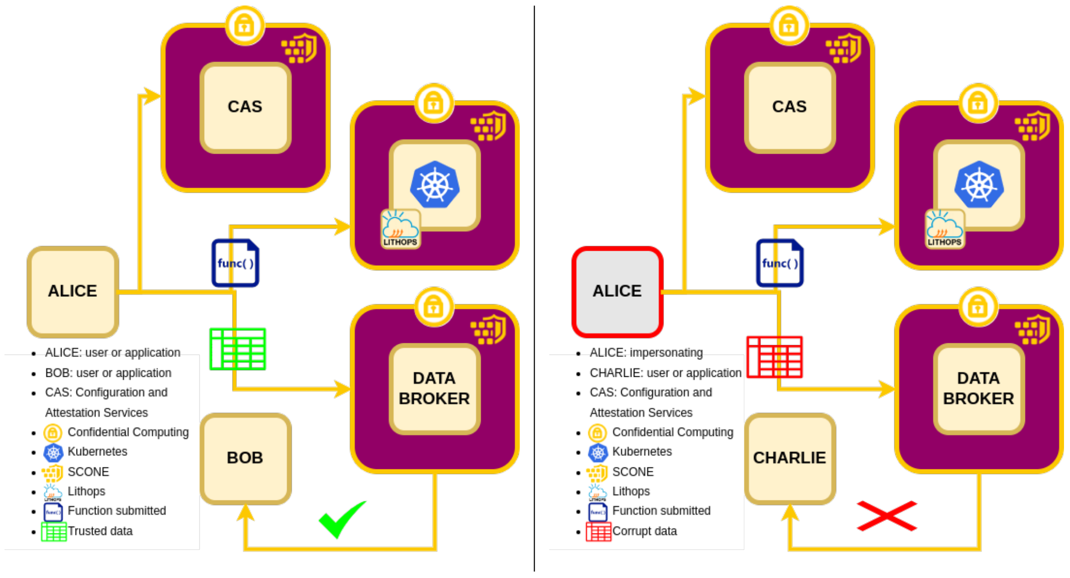


Figure 6: Left: permitted communication. Right: not permitted communication.

The alternate scenario above can be inhibited by the use of a versatile IAM like Keycloak. It is ported to SCONE, therefore can run in confidential computing mode – CIAM, using the same set of policies provisioned by CAS to the other applications, hence trustworthy among the entities of the domain. Alice has only the role "*genomics-adenine-analysis*", which grants her communication with Bob. Charlie does not recognize such role as valid, he only accepts to talk to people who have the role "*genomics-thymine-analysis*". All the three users are managed by Keycloak and have each of them the specified roles accordingly to their needs and responsibilities.

The security improved scenario brought by Keycloak can inhibit the last step of the attack. The Data Broker should be able to filter invalid communication requests, based on the roles the source and destination matches. Replaying the scenes above, with this new character (Keycloak), in summary, this is the workflow: *1)* Alice will login to Keycloak using her private password and issue an *Access Token* – AT containing her roles. *2)* Alice will submit the produced payload resulting from Lithops processing to the Data Broker, with her AT in HTTP header. *3)* The Data Broker will check whether Alice's roles present in the AT are allowed by the recipient. *4)* If so, the Data Broker will request a *Validation Token* – VT to Keycloak based on the AT received, if not, the communication is aborted. See Fig. 7. This setup places another layer of protection on the operations, by establishing a direct *who to whom* relation; it is enforced by the parties, especially the submitting one, having to authenticate themselves with their private user and password in order to consume services. *Please refer to the document "NEARDATA D2.1 Initial Architecture Specifications, A.2.2." concerning Keycloak API.*

4.7 Governance and Policy Board

The Data Broker component will ensure secure data access and the orchestration of dispersed data sources by leveraging TEEs and adopting federated learning architectures, all within a multi-stakeholder framework. This strategic approach places a strong emphasis on collaboration among multiple stakeholders, ensuring that each entity retains control over its own data while actively contributing to the collective improvement of models.

In many applications, multiple stakeholders collectively govern the system. In certain domains, there is a reluctance to delegate decision-making authority unilaterally to a single entity. For instance, the power to determine which applications gain access to specific secrets should not rest solely with a single administrator. Instead, such critical decisions require the involvement of a group composed of representatives from various stakeholders. This collaborative governance model ensures a bal-

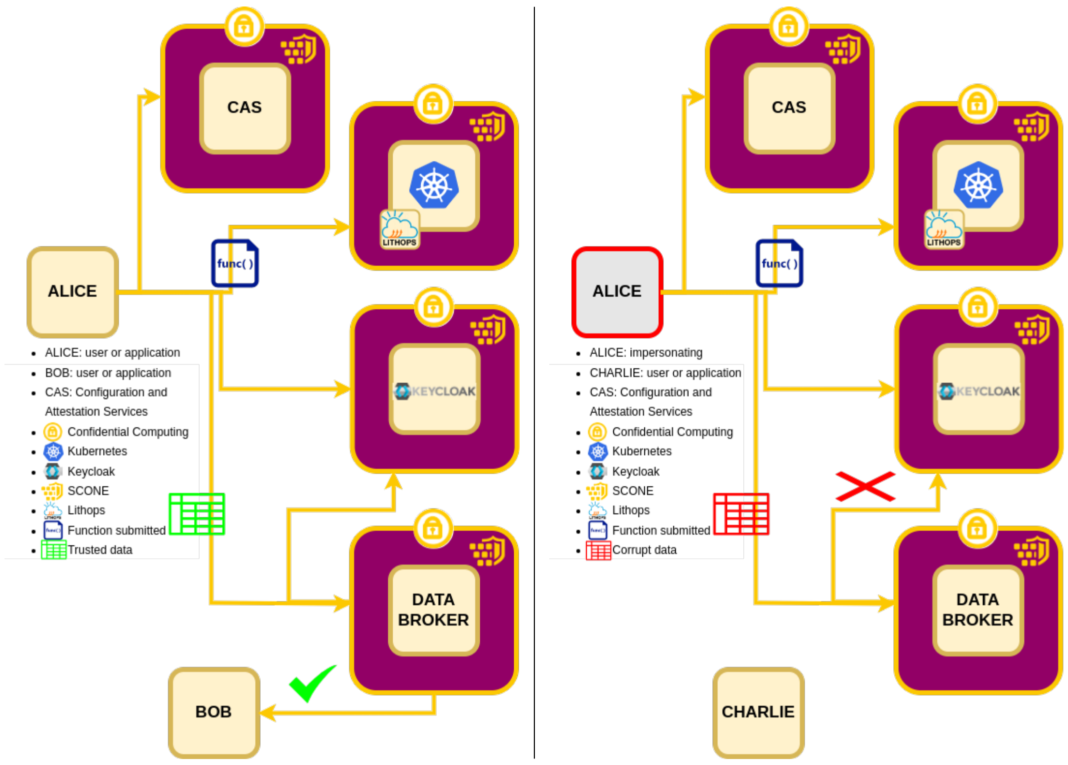


Figure 7: Left: permitted communication. Right: not permitted communication.

anced and inclusive approach, where decisions are made collectively, taking into account the diverse perspectives and interests of the involved entities. This multistakeholder framework enhances transparency, fairness, and the overall effectiveness of decision-making processes within the application ecosystem.

The CAS component of the Data Broker facilitates sophisticated governance mechanisms, where the creation and modification of policies require approval from a designated policy board. This board may comprise both human members and automated policy checkers, referred to as programs. The approval process involves the policy board voting through the signing of policy changes. Only when a sufficient set of signatures is appended to the policy will the creation or modification be accepted, with the policy explicitly specifying which signatures are deemed sufficient.

Centralization of the declaration of signatories for a policy is possible, allowing governance rules to be imported from other policies, termed as the import of policy fragments. This functionality empowers the system to efficiently manage and implement governance rules.

Integrated access control is an inherent feature of each security policy. Within the policy, specifications are made regarding the individuals or entities authorized to access it. The Data Broker enhances governance by stipulating the specific stakeholders whose agreement is essential for the modification of existing policies or the creation of new ones. This ensures a structured and collaborative approach to policy management within NEARDATA.

4.7.1 Access Control

In the context of a Data Broker, any operation on a session description requires explicit permission. If the entity seeking a specific operation lacks the necessary authorization, the request will be unsuccessful. To manage these permissions, the access_policy keyword is employed, enabling the specification of lists of entities permitted to carry out particular operations. The following operations can be controlled through the access_policy:

1. **Read:** Permission to access and read the session description, excluding any confidential information generated by the system.

2. **Update:** Authorization to modify or update the session.

3. **Create_sessions:** Permission to generate new sessions within the designated namespace of the current session. If omitted, entities listed under the update operation are automatically granted the ability to create sessions.

Granting permission to a specific entity for these operations involves the inclusion of their client certificate public key in the list of authorized entities. The client certificate, containing this key, is then utilized to establish a secure connection, with TLS ensuring the possession of the corresponding private key. The system employs key-based authentication, enhancing security and allowing for certificate renewal without the risk of users being locked out of sessions upon certificate expiration.

The system supports various values for specifying access permissions in addition to public certificates. These values include:

- **Public Key Hash of a Certificate:** Calculated using the SCONE CLI, where *scone self show* displays the hash of the CLI client identity, and *scone cert show-key-hash* "path_to_certificate_file_in _pem _format" shows the key hash for any certificate file.

- **CREATOR Keyword:** Grants access to the creator of the policy, identified by the public key of the TLS client certificate used during the session's creation.

- **ANY Keyword:** Allows access to any entity. If *ANY* is specified, no other entries in the list for that operation are permitted.

- **NONE Keyword:** Denies all requests for a specific operation. If *NONE* is specified, no other entries in the list for that operation are allowed.

- **SCONE::secretname Placeholder:** Dynamically uses the value of a secret with the given name *(secret-name)* during permission evaluation. The replaced value must be either *CREATOR (ASCII)*, a certificate key hash (ASCII), or a certificate (X.509) as defined above. Explicit secrets, generated secrets, and imported secrets are supported. When referencing X.509 certificates, specifying a format suffix is not necessary. It will be ignored if the mentioned secret does not exist, cannot be read, or has an incompatible value. An error message will be shown on unsuccessful authentication attempts.

- **Signer: <public key>:** Represents the public key of a session signer. Signing sessions serve as an alternative to TLS client certificates during session upload. *scone self show-session-signing-key* displays the CLI signer identity. The public key is Base58check-encoded (BITCOIN alphabet), with a version of 10, and the encoded key must be an Ed25519 key.

- **Require-all or Require-at-least-X:** rules are specifically designed for use with signers in the context of session creation. These rules acknowledge that a session can be established using either one TLS client certificate or one or more signatures. Consequently, specifying multiple certificates simultaneously is not meaningful.

By default, the access policy is defined as follows:

```

access_policy:
 read:
  - CREATOR
 update:
  - CREATOR
 create_sessions:
   <same as update>

```

These versatile options for specifying access permissions provide a comprehensive and flexible approach to managing and controlling operations on session descriptions within the Data Broker system.

4.7.2 Policy Board

In a multi-stakeholder environment, where collaboration involves multiple entities with distinct credentials, relying on a single credential for controlling access to a session is deemed insufficient. To address this, the governance system introduces a robust mechanism that enables multi-credential access control. This is achieved through the use of require-all and require-at-least-X rules.

The governance rules, specifically *require-all* and *require-at-least-X*, play a crucial role in the *update* and *create_sessions* access policies within the governance framework. However, it's important to note that these rules do not apply to the read access policy. The rules, when used, are designed to be employed with signers. Even though it is technically possible to specify certificates or certificate key hashes, it is emphasized that a session can be created using precisely one TLS client certificate or one or more signatures. Therefore, requiring more than one certificate simultaneously doesn't align with the typical session creation process.

```

access_policy:
 read: NONE
 update:
  require-at-least-1:
   - signer: $owner
   - require-all:
      - require-at-least-2:
          - signer: $voter1
          - signer: $voter2
          - signer: $voter3
      - require-all:
          - signer: $veto_member1
          - signer: $veto_member2
```

This configuration enforces governance rules within the updated access policy. Specifically, it requires at least one approval from the $owner. Additionally, it introduces nested require-all rules. The first nested requirement specifies that at least two out of three voters ($voter1, $voter2, $voter3) must approve. The second nested require-all ensures that both veto members ($veto_member1, $veto_member2) must provide their approval.

This example demonstrates a sophisticated governance structure where multiple stakeholders with different roles must provide their approval for certain operations. It also showcases the flexibility of the rules, allowing for nested configurations to accommodate complex governance scenarios.

It's worth noting that this example provides a way for the $owner to overrule all other members, adding a layer of authority to the governance structure.

5 First Release of the DataBroker

This section describes the components of Data Broker. For the components description, we will use the Software Design Specification (SDS) standard (IEEE Standard 1016) in order to better describe its purpose, function and where it sits in the highlever architecture. In particular, this section focuses exclusively on the components which are control data access and provide security measures within NEARDATA.

The section commences with an overview of the runtime, progressing to the integration of the Data Broker with the CAS service. Subsequently, essential CLI tools are introduced for the transformation of a native application into a confidential ones. A detailed discussion is dedicated to each of these components. To structure information for every component within the runtime environment, the document employs the following template. This inclusion ensures the self-containment of the document.

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.
Needed improvement	Description of the needed improvements of this tool with regards the NEARDATA project, in order to fulfil the user requirements and to build the runtime environment
Implemented Improvements for the First 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 7: First 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 two classes of hardware support for confidential computing available. The current mechanisms can be classified as follows:

- one can encrypt a Virtual Machine (VM) in which the application is executing, and
- one can encrypt each individual service, i.e., process inside of an enclave

One has to be careful regarding the security guarantees because the use of an encrypted VM does not necessarily mean that a root user of the host does not have access to the VM. For example, in current AMD CPUs without Secure Nested Pages (SNP), the hypervisor could break the confidentiality and integrity of an application[3]. Also, the consistency of memory pages is not protected by AMD CPUs, i.e., one could replace a memory page by an older version: This would properly encrypt data but would break consistency. However, in Intel SGX enclaves, it is guaranteed that applications always read the most recent data that was written. Hence, it is important to review the different guarantees each CPU vendor provides with regards to the desired properties when applying the confidential compute paradigm.

In the following, we will put a focus on enclaves such as those provided through Intel SGX. However, we will also discuss our future extension to support vendors other than Intel CPUs such as AMD, ARM etc.

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.

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

[3]https://www.amd.com/system/files/TechDocs/SEV-SNP-strengthening-vm-isolation-with-integrity-protection-and- more.pdf

- 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.

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.

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.

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.

One of the arguments of using encrypted VMs (instead of enclaves) is that it simplifies the protection of existing applications. In reality, this is of course not that simple since a user

- has to encrypt files of the VM,
- one has to ensure that the VM learns the filesystem encryption key but only if neither the operating system nor the application was not modified and it is executed in an encrypted VM
- one has to ensure that the service in the different VMs do attest each other, and
- one has to provision secrets etc

Using the enclave-based approach, it is possible to transform existing container images into confidential container images in a single step (as we will explain in the Sconify section below). The SCONE framework we will use as a foundation in NEARDATA provides furthermore a policy language that permits to define on how to provision secrets and how to attest applications, i.e., addresses all these issues using a simple, YAML-based policy shown previously.

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.

Features	Intel SGX SDK	SCONE Platform
SLA: Startup times	Slow	Efficient startup/attestation
SLA: Scheduling	-	SLA-based scheduling
SLA: Efficiency	Many enclave exits	Reduced enclave exits
Security: CVEs	CVE handling by application	CVEs addressed by platform
Security: policy	No policy support	Advanced-policy support
Security: platform	-	Integrated OS and Application Sec.
Security: Side-channel	No protection	Side-channel protection
Monitoring: SLA	-	SLA-based monitoring
Monitoring: SGX	-	SGX-resources & scheduling
Encryption at rest / in transit	Source code changes required	No source code changes
Encryption at use	Source code changes required	No source code changes
Attestation	Explicit code required	Automatic by SCONE
Key Provisioning	Explicit code required	Automatic by SCONE
CI/CD Integration	-	Modern IDE
Languages	-	Modern IDE
Portability	Intel SGX-specific	(eventually other CPUs)
TCO	Higher	Lower

Table 8: Intel SGX SDK vs. SCONE

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.

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 provides applications with secrets in a secure

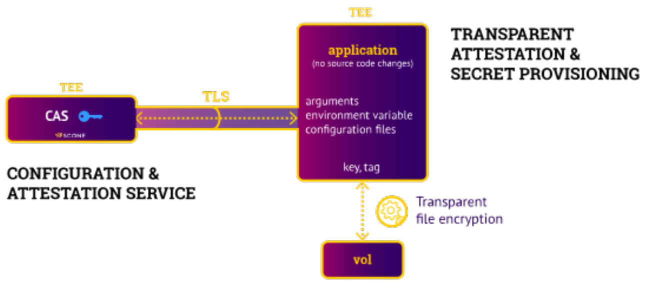
Identification	SCONE Runtime (SCONE)
Type	A service that runs alongside the application.
Purpose	The SCONE runtime ensures that an application runs in a trusted execution environment. It performs preparation as well as attestation of code and data to run in an enclave.
Function	The runtime manages the complete deployment of an executable/process in a so-called enclave and performs the following tasks: 1. creation/allocation of enclave memory 2. loading program code and data into enclave memory 3. performing attestation, i.e., creating measurement over code and data and 4. verifying if this is correct 5. performing configuration of the application 6. provisioning secrets 7. providing network as well as file system encryption/shielding
High level Architecture	The following image describes the high-level architecture of the SCONE Runtime, including external dependencies.  TEE application (no source code changes) arguments environment variable configuration files key, tag TRANSPARENT ATTESTATION & SECRET PROVISIONING TEE CAS SCONE TLS CONFIGURATION & ATTESTATION SERVICE Transparent file encryption vol The SCONE runtime runs alongside with the actual process, hence it has some start code to first allocate enclave memory, load the application code and data into the enclave and then connects to CAS in order to verify if the hash consisting of application code and data matches the expected measurement (MREnclave). It furthermore provisions the application with configuration files as well as secrets such as certificates and keys.
Dependencies	The SCONE runtime requires access to CAS in order to perform attestation as well as secret provisioning.
Interfaces	The SCONE runtime provides three modes of execution. HW, where the process runs in a TEE and will abort if the hardware support is not given, SW mode which can be used for simulation purposes or if only features like network and file system encryption are needed. The AUTO mode refers to the mode where the executable will leverage TEE support if available but will run in simulation mode rather than aborting the process execution. The mode is controlled through environment parameters when theapplication/process is being launched.
Data	The SCONE runtime uses SCONE policies as well as Environment variables to configure the application correctly.
Needed improvement	Support for CPU vendors other than Intel such that the component can run also on edge devices such as ARM. This will be achieved through compilation using the SCONE Cross compiler which will be also implicitly used by the SCONE Sconify CLI Tool.
Implemented Improvements for the First Release	The first release of the SCONE runtime regarding NEARDATA includes the following improvements: 1. Created first draft/implementation of the network shielding layer 2. Integration with latest version of Data Broker CAS component
Release Version & Repository	The first release version of the SCONE runtime for NEARDATA can be downloaded from the Docker registry: `https://sconedocs.github.io/registry/`

Table 9: SCONE Runtime

fashion. This is typically a problem if one wants to run TensorFlow and configures an AI application to encrypt its resulting model stored locally. To do so, the Python code requires a key to decrypt and encrypt its files. This key can be stored in the Python file itself or some configuration file but this configuration file cannot be encrypted since Python would need a key to decrypt the file. SCONE helps developers to solve such configuration issues in the following ways:

- **secure configuration files** - SCONE can transparently decrypt encrypted configuration files, i.e., without the need to modify the application. It will give access to the plain text only to a given program, like Python. No source code changes are needed for this to work.
- **secure environment variables** - SCONE gives applications access to environment variables that are not visible to anybody else - even users with root access or the operating system. This is an important feature when considering the Python example from above. The user can pass passwords via environment variables like MODEL_PASSWORD to the Python process. We need to protect these environment variables to prevent unauthorized access to the secret and the model encrypted by this secret.
- **secure command line arguments** - Some applications might not use environment variables but command line arguments to pass secrets to the application. SCONE provides a secure way to pass arguments to an application without other privileged parties, like the operating system, being able to see the arguments as shown in Figure 8

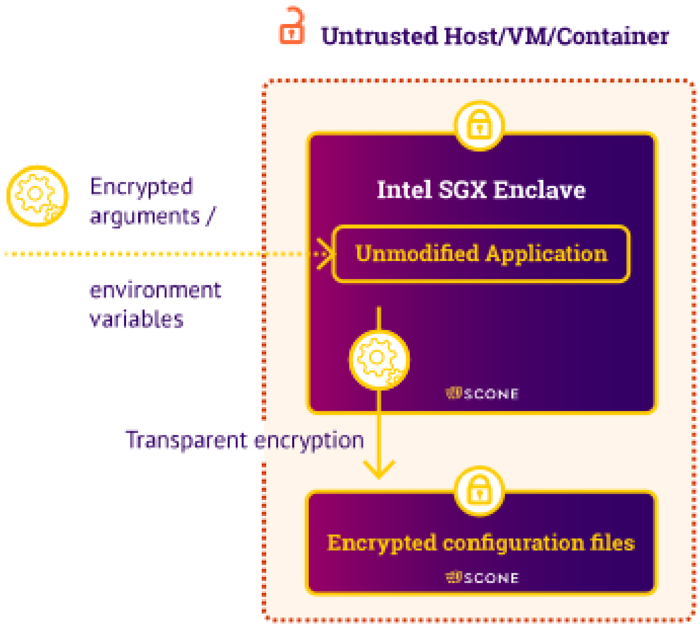


Figure 8: Program arguments as well as environment variables are provided through CAS.

SCONE verifies that the correct code is running before passing any configuration info to the application. To ensure this, SCONE provides a local attestation and configuration service: this service provides only the code with the correct signature (MrEnclave) with its secrets: certificates, arguments, environment variables and keys. It also provides the application with a certificate that shows that the application runs inside an enclave as depicted in Figure 9. Note that, this can be done completely transparently to the application, i.e., no application source code changes are required: the encrypted certificate can be stored in the file system where the application expects its certificates. Note that, for debugging and development purposes, an end user can run code inside of enclaves without attestation.

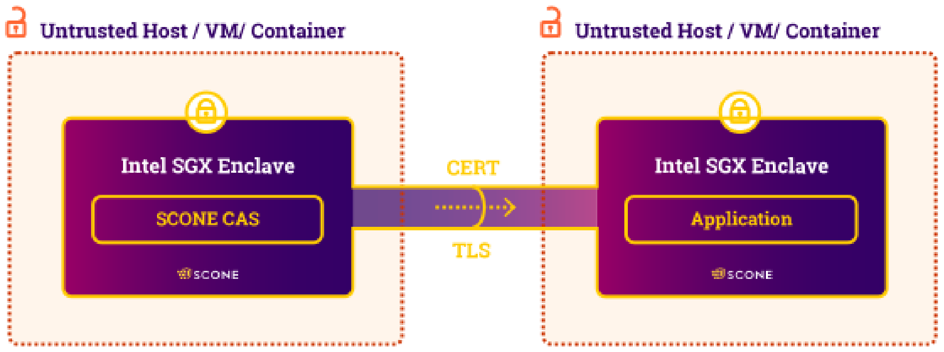


Figure 9: Example who one entity is being authenticated using TLS.

Two applications can ensure that they run inside enclaves via TLS authentication. In this way, we can ensure that the client certificate and the server certificate was issued by the SCONE CAS, i.e., both communication partners run inside of enclaves and have the expected MrEnclave as shown in Figure 10. We leverage this feature in federated learning to attest if collaborators are legitimate to communicate with aggregators and vice versa. Hence, our Data Broker component can precisely control which parties are allowed to establish communication in order to exchange either streaming data such as Pravega or data stored at rest, e.g. in S3 compatible stores.

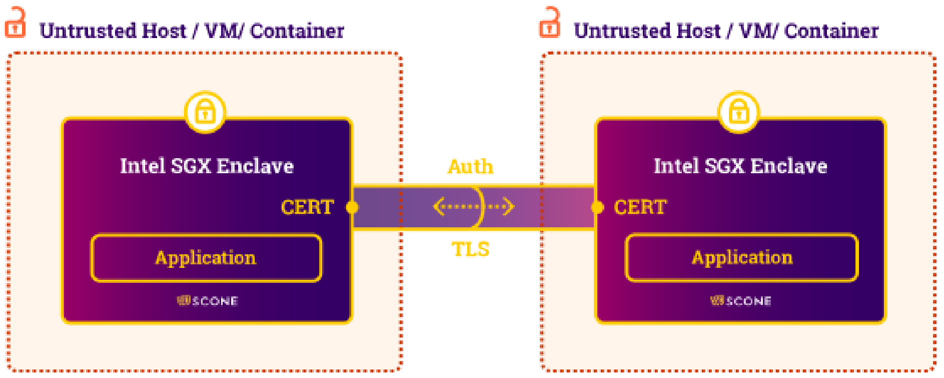


Figure 10: Example about how two entities authenticate each other using TLS - mutual authentication

An adversary with root access can read the memory content of any process. In this way, an adversary can gain access to keys that an application is using, for example, the keys to protect its data at rest such as the patient data used in the federated learning example. SCONE helps to protect the main memory:

- no access by adversaries - even those who have root access,
- no access by the operating system - even if compromised,
- no access by the hypervisor - even if compromised,
- no access by the cloud provider, and
- no access by evil maids - despite having physical access to the host.

To provide full protection, encryption keys must be protected as well. However, in many installations, one does not want humans to be able to see these encryption keys. Hence, one can generate keys and stores in SCONE CAS. SCONE also supports the integration with keystores like Vault. SCONE can run Vault inside of an enclave to protect Vaults secrets in main memory.

For those applications, e.g., memcached or Zookeeper that do not support TLS out of the box, SCONE can transparently add TLS encryption to TCP connections, i.e., the connections are terminated inside of the enclave. In this way, the plain text is never seen by the operating system or any adversary as shown in Figure 11. Note that one should not use an external process for TLS termination such as stunnel.

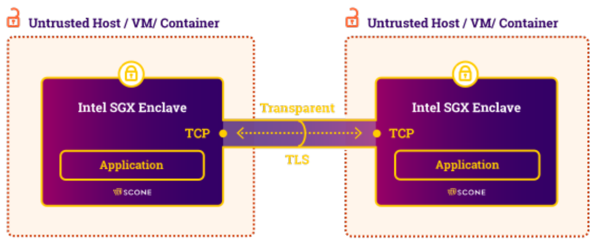


Figure 11: Example of transparent network encryption. Connections are terminated inside the enclaves.

SCONE furthermore protects the integrity and confidentiality of files via transparent file protection. This protection does not require any source code changes. A file can either be integrity-protected only (i.e., the file is stored in plain text but modifications are detected) or confidentiality- and integrity-protected (i.e., the file is encrypted and modifications are detected) as shown in Figure 12.

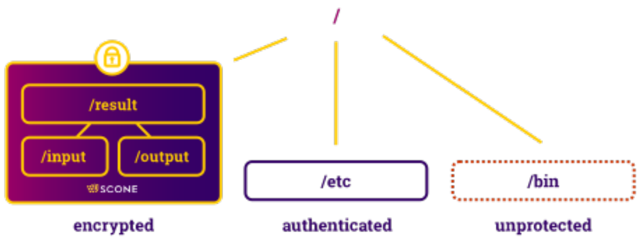


Figure 12: SCONE file system shield - left volume that is encrypted, an integrity protected but not encrypted volume (middle), right no protection at all.

5.2.2 SCONE Cross Compiler

Identification	SCONE Cross Compiler
Type	A CLI tool that compiles source code into a confidential application
Purpose	The SCONE cross compiler allows application developers to compile application source code such that the resulting binary contains additional instructions to leverage TEEs such as Intel SGX. Besides the additional instructions, it also adds the starter code needed to copy the application code and data into the enclave memory as well as launching the enclave.
Function	The cross compiler performs compilation of source code - several languages are supported as to date: C C++ Fortran Go Rust Python* *The python interpreter is written in C hence, it is compiled with SCONE's C cross compiler. Then we use the file system protection shield to protect the python code itself with regards to confidentiality as well as integrity. Hence, there is no cross compiler for Python. However, the sconify tool performs those two steps, i.e., performing the cross compilation as well as creating the volumes including the encryption of the python source files such that everything is protected.
High level Architecture	The cross compiler is a CLI tool, hence it does not interact with other services.
Dependencies	The cross compiler is packaged as a Docker image, hence all dependencies are included in the Docker image. However, currently the cross compiler supports only compilation of applications that depend on musl or glibc as libc implementation.
Interfaces	The cross compiler is launched like a regular compiler on the command line.
Data	No date required.
Needed improvement	As a next step, we plan to integrate support for other CPU vendors such as ARM. The analysis of our current code base reveals that several functions were written in Assembly which must be ported to ARM. An alternative is to replace those code parts with less performant implementation based on C where applicable. Furthermore, the build pipeline must be adjusted.

Table 10: SCONE Cross Compiler

SCONE supports running applications written in common programming languages inside of Intel SGX enclaves without source code changes. These languages include compiled languages like C, Rust, C++, GO, and Fortran and interpreted / just-in-time languages like Python and Java. For compiled languages, our recommend approach to run an application with SCONE is as follows:

- Use of precompiled binary: For many common applications like nginx and memcached, we already support a curated image image on registry.scontain.com.
- Cross-compile: developers can cross-compile applications with the help of the SCONE cross-compilers
- No Cross-Compilation: users can run native Alpine-Linux applications inside of enclaves without recompilation.

While SCONE supports the execution of programs without recompilations for Alpine Linux, the recommended approach is to always cross-compile: To benefit from SCONE's functionality, the interface to the operating system needs to be replaced, i.e., libc. Hence, this requires not only to provide the same version of libc but also to ensure that all bits are represented in the same way as in the native libc. This is difficult to achieve and better left to the compiler. For stability, the recommended approach is to cross-compile as the compiler checks that all the dependencies have the matching versions, all data types are bit compatible and includes the correct libraries statically in the binary. In this way, an application will have a unique and known MrEnclave.

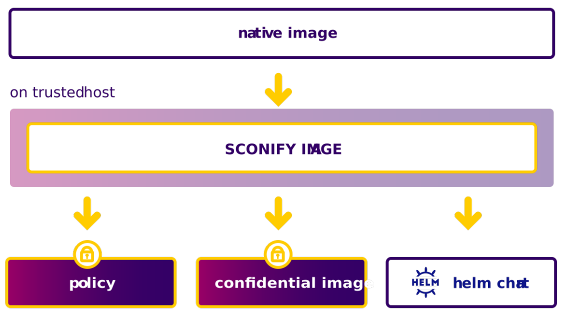
Identification	SCONE Sconify Tool
Type	A CLI tool that turns a native Docker image into a confidential image that uses the SCONE runtime
Purpose	The SCONE sconify tool modifies existing Docker images in such a way that the applications processes launched inside the container/image will utilize the SCONE runtime, i.e., can run in enclaves and hence benefit from TEE support.
Function	The sconify tool re-links the SCONE libc. However, several constraints must be first met which the tool tries to do automatically: . all executables must be compiled with the PIE option (position independent code) . original images must be either musl or glibc-based . language support is enforced by the tool: C, C++, Python, Java, Go and Node.js . Go support is currently limited to binaries compiled with gcc-go (this must be detected as well) . for Node.js, the tool replaces the Node.js interpreter with the one of the SCONE curated Node.js (for other interpreter, such as Python we sconify the original interpreter) Additional steps the tool performs: . It determines which libs the application relies on. Since the tool is built with dlopen=1, sconify has a few default places it includes automatically, such as usrlib, lib, etc. . Encryption of the file system, i.e. for every file needed by the application automatic detection of shared dependencies in order to copy them over Last step - Manifest generation: . generation of Dockerfiles for the target image . generation of SCONE policies and submitting them to an attested CAS . generation of helm charts for the application
High level Architecture	The sconify tool is a CLI tool, hence it does not interact with other services.  native image on trustedhost SCONIFY IMAGE policy confidential imag HELM helm chat
Dependencies	The sconify tool is packaged as a Docker image, hence all dependencies are included in the Docker image.
Interfaces	The sconify tool is launched like a regular CLI tool on the command line.
Data	No data required.
Needed improvement	Extend support for Go-compiled programs and libc implementations other than GLibc and musl.

Table 11: SCONE Sconify Tool

5.2.3 SCONE CAS (Configuration & Attestation Service

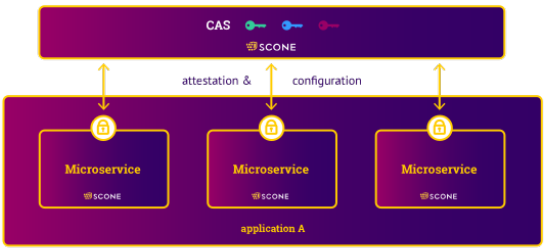
Identification	SCONE CAS (CAS)
Type	A service that provides remote attestation as well as configuration and secret provisioning for confidential applications.
Purpose	The SCONE CAS can be envisioned as a database and REST-based service that contains secrets as well as provides several services such as code attestation and secret provisioning.
Function	The CAS performs the following tasks: . code attestation - local or remote . configuration provisioning . management of SCONE policies . perform configuration of the target application
High level Architecture	CAS (the Configuration and Attestation Service) will be contacted by an application during the launch procedure in order to verify if the hash consisting of application  CAS SCONE attestation & configuration Microservice SCONE Microservice SCONE Microservice SCONE application A code and data matches the expected measurement (MREnclave). It furthermore provisions the application with configuration files as well as secrets such as certificates and keys.
Dependencies	CAS (the Configuration and Attestation Service) needs to be able to establish a connection to the Intel Attestation Service (IAS) in order to verify its own legitimacy.
Interfaces	CAS provides a restful interface (REST) for the management of SCONE sessions, i.e., the configuration of applications.
Data	CAS uses SCONE policies/sessions provided in YAML syntax.
Needed improvement	Recompilation/redistribution with ARM, etc. support once the SCONE cross compiler supports it.

Table 12: SCONE CAS (Configuration Attestation Service)

In the following, we will provide more details about Data Broker functionality as well as its guarantees it provides with regards to security. First we quickly present the purpose of CAS followed by the different functionalities such as key generation and management, and access control.

Data Broker integrates with SCONE CAS manages the secrets - in particular, the keys - of an application. The application is in complete control of the secrets: only services given explicit permission by the application's policy get access to keys, encrypted data, encrypted code and policies.

Key generation. SCONE CAS can generate keys on behalf of an application. The generation is performed inside of a trusted execution environment. Access to keys is controlled by a security policy controlled by the application. Neither root users nor SCONE CAS admins can access the keys nor the security policies. So far, SCONE CAS runs inside of SGX enclaves.

Isolation. Users can run their own instances of SCONE CAS, i.e., one can isolate the secrets of different users and the secrets of different applications.

Secure key and configuration provisioning without the need to change the source code of applications: secrets, keys, and configuration parameters are securely provisioned via command line arguments, environment variables and via transparently encrypted files.

Access control. To modify or read a policy, a client needs to prove, via TLS, that it knows the private key belonging to a public key specified in the policy. SCONE CAS grants - without any exception - only such clients access to this policy. The client's access to a private key is typically also controlled by a policy - possibly, even the same policy. Note that, only after a successful attestation, a client can get access to its private keys.

Management. The management of SCONE CAS can be delegated to a third party. The confidentiality and integrity of the policies and their secrets are ensured by CAS itself. Since the entity creating a policy has complete control over who can read or modify this policy, no admin managing the CAS can overwrite the application's access control to a policy.

Encrypted Code. One can create images with encrypted Python code or Java or JavaScript or C# or any other JIT or interpreted code on a trusted host. Alternatively, this code could als be generated inside of an enclave. One can transparently attest and decrypt the code inside of an enclave. This can be done without the need to change the Python engine or the Java etc. virtual machine. Note that, SCONE CAS attests both the Python engine as well as the Python code.

5.3 Network Security

In this section, we will briefly present the network shield, a software layer integrated into the SCONE runtime and used by our Data Broker, to (a) transparently apply end-to-end encryption to connections established by protected services, preserving confidentiality and integrity of data on the network, (b) mutually authenticate endpoints while taking remote attestation into account to ensure that they run an unmodified software in a secure environment, and (c) filter connections according to user-supplied communication authorization rules to prevent leaking data to unauthorized entities.

Our design and implementation is based on standard protocols such as TLS, which we integrate by intercepting and reprocessing I/O system calls issued by the service application. We furthermore provide a X.509 v3 public key infrastructure to couple authentication with the software attestation and secret provisioning capabilities of the Configuration and Attestation Service (CAS) and we plan to extend the CAS' configuration interface to allow granting network access permissions on the basis of individual ports, including the ability to authorize communication with external services or legacy clients. For applications such as nginx or Apache httpd, our implementation based on mbedtls [4] achieves 76% to 100% of the throughput of application-provided TLS for persistent connections at a very small latency increase.

Most protected application services running inside an enclave will need to securely exchange data with other services in order to operate. We propose a system revolving around transparent encryption to ensure the confidentiality and integrity of a service's network communication, while simultaneously enforcing controlled access by mutually authenticating authorized remote services. The transparent encryption system will be implemented as part of the SCONE runtime. An overview of the system's control flow is shown in Figure 13 .

[4] https://github.com/ARMmbed/mbedtls

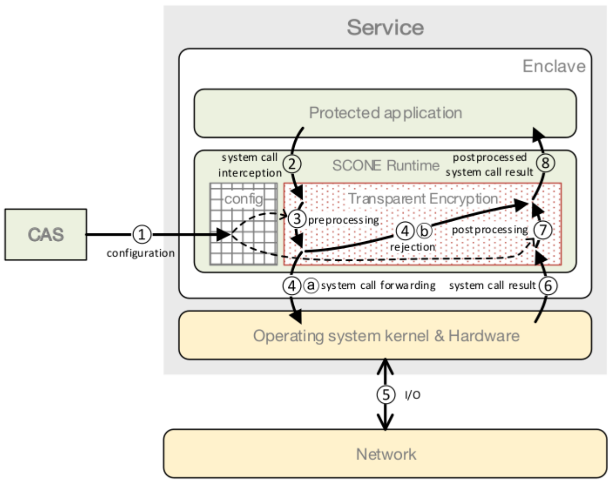


Figure 13: Control flow of transparent encryption system operations.

1. The runtime's configuration is provisioned by CAS - currently through environment variables. This includes an initial configuration phase and subsequent, periodic updates. The configuration contains information that allows the service to identify itself to remote services, that determine the mode of operation (protected, unprotected, opportunistic or refuse based on network addresses of either local or remote peer and information to authenticate authorized remote services).

2. System calls related to network communication issued by the runtime-enabled application running inside the enclave (subsequently referred to as "protected application" or just "application") will be intercepted using the SCONE runtime's capabilities before they are handled by the kernel.

3. The arguments of intercepted system calls will be preprocessed, without imposing additional requirements on the protected application. The Network Shield is API-preserving and thread-safe, as the system call interface must always be thread-safe. Preprocessing may involve access control mechanisms or payload encryption. It may use previously received configuration entries.

4. 4a) If preprocessing succeeds, the system call will be forwarded with modified arguments to the untrusted OS kernel.

   (4b) If preprocessing fails, the system call will be rejected instead. System calls whose arguments pose a risk of leaking confidential information or are incompatible with the system's implementation are subject to rejection.

5. The OS kernel will (potentially asynchronously) exchange data with the network based on the previously issued system calls.

6. The kernel propagates the system call result back to the SCONE runtime.

7. The received result will be post-processed. This may involve validation or payload decryption. Post- processing may also use the runtime's configuration.

8. The postprocessed system call result or rejection error code from (4b) is propagated back to the protected application.

Now that we have established how the network shield's transparent encryption system is able to process system calls, we can use this as a building block to define how socket-based communication can be protected. To achieve our data confidentiality and integrity goals, we require an authenticated encryption scheme. Additionally, service enclaves must authenticate each other to prevent any unauthorized accesses. For this purpose, we use TLS, which offers a standardized and well-researched protocol solution for both problems.

We utilize its X.509v3 certificate support for mutual authentication. Once a protected server application opens a listening socket, the network shield will transparently start a TLS server in its place. When a client connects, a TLS handshake will be performed to authenticate the client and setup an encrypted session. Connections from unauthorized clients will be terminated immediately, prior to being able to interface with the protected application. Data sent by the server to authorized clients will be encrypted as part of the preprocessing before being forwarded to the untrusted OS kernel, and data received will be decrypted before being provided to the application. A client socket will behave similarly, starting a TLS client which connects to the remote TLS server instead.

This approach works well for sockets of stream- and connection-oriented network protocols such as TCP, but it is not applicable to sockets of message-oriented connectionless protocols like UDP. While our primary focus lies on TCP as the network protocol most commonly used in service interactions, we propose a modified version of the shield for the second use case, using DTLS in place of TLS and grouping sent and received messages into virtual connections protected by a single DTLS session based on network address and timing information.

TLS session configuration shall follow cryptography best practices, such as using strong, non-deprecated cipher suites with a security level of at least 128 bit, providing forward secrecy by the use of Elliptic—curve Diffie-Hellman Ephemeral (ECDHE) key exchanges and disabling features known to cause adverse effects on security (such as TLS compression or insecure renegotiation).

6 Performance Evaluation

In this chapter, we will present various performance evaluations we executed in order to assess the overhead introduced through our approach.

The federated learning example is executed within SGX enclaves using SCONE's network shield, which is set using the Data Broker's CAS component.

We assess the performance of the confidential federated learning application by analysing the impact of SCONE's network shield, configured through the Data Broker's CAS component, on execution time. We compare this with both the Vanilla (Intel/Non-SGX) environment and the SCONE (Intel SGX/hardware mode) setup, with a particular focus on the execution time of federated learning. Furthermore, we evaluate the effects of optimizations on the startup latency of functions within the FL application architecture (***refer D5.1 First release of KPI benchmarks in all use cases and data connector libraries***).

Furthermore, we will present the runtime-related performance results followed by an evaluation of the network shield we introduced in NEARDATA.

6.1 Runtime Security

In this section, we present the evaluation results of SCONE applied to TensorFlow based on both micro-benchmarks and macro benchmarks with a real-world deployment.

6.1.1 Experimental Setup

Cluster setup. We used three servers with SGXv1 support running Ubuntu Linux with a 4.4.0 Linux kernel,equipped with an Intel© Xeon© CPU E3-1280 v6 at 3.90GHz with 32 KB L1, 256 KB L2, and 8 MB L3 caches, and 64 GB main memory. These machines are connected using a 1 Gb/s switched network. The CPUs update the latest microcode patch level.

In addition, we used a Fujitsu ESPRIMO P957/E90+ desktop machine with an Intel© core i7-6700 CPU with 4 cores at 3.40GHz and 8 hyper-threads (2 per core). Each core has a private 32KB L1 cache and a 256KB L2 cache while all cores share an 8MB L3 cache. **Datasets.** We used two real world datasets: (i) Cifar-10 image dataset [29] and (ii) MNIST handwritten digit dataset [30]:

- Cifar-10: This dataset contains a labeled subset of a much larger set of small pictures of size 32x32 pixels collected from the Internet. It contains a total of 60,000 pictures. Each picture belongs to one of ten classes, which are evenly distributed, making a total of 6,000 images per class. All labels were manually set by human labelers. Cifar-10 has the distinct advantage that a reasonably good model can be trained in a relatively short time. The set is freely available for research purposes and has been extensively used for benchmarking machine learning techniques [31, 32].

- MNIST: The MNIST handwritten digit dataset[31] consists of 60000 28 pixel images for training, and 10000 examples for testing.

6.1.2 Methodology

Before the actual measurements, we warmed up the machine by running at full load with IO-heavy operations that require swapping of EPC pages. We performed measurements for classification and training both with and without the file system shield. For full end-to-end protection, the file system shield was required. We evaluate TensorFlow with two modes: (i) hardware mode (HW) which runs with activated TEE hardware and (ii) simulation mode (SIM) which runs with simulation without Intel SGX hardware activated. We make use of this SIM mode during the evaluation to evaluate the performance overhead of the Intel SGX and to evaluate TensorFlow when the EPC size is getting large enough in the future CPU hardware devices.

6.1.3 Micro-benchmark: Remote Attestation and Keys Management

In TensorFlow, we need to securely transfer certificates and keys to encrypt/decrypt the input data, models and communication among worker nodes (in a distributed training process). To achieve this

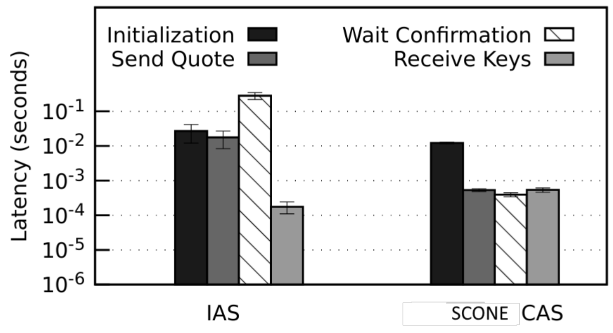


Figure 14: The attestation and keys transferring latency comparison between TensorFlow with the traditional way using IAS.

security goal, we make use of the SCONE CAS component which attests TensorFlow processes running inside enclaves, before it transparently provides the keys and certificates to encrypt/decrypt input data, models, and TLS communications. Note that the advantage of using CAS over the traditional way using IAS (Intel Attestation Service) to perform attestation is that the CAS component is deployed on the local cluster where we deploy TensorFlow.

Figure 14 shows the break-down latency in attestation and keys transferring of our component CAS and the method using IAS. The quote verification process in our CAS takes less than 1ms, whereas in the IAS-based method is 280ms. In total, our attestation using CAS (17ms) is roughly 19× faster than the traditional attestation using IAS (325ms). This is because the attestation using IAS requires providing and verifying the measured information contained in the quotes [30] which needs several WAN communications to the IAS service.

6.1.4 Macro-benchmark: Inference Classification Process

We evaluate the performance of TensorFlow in SCONE in real-world deployments. We present the evaluation results of TensorFlow in detecting objects in images and classifying images using pre-trained deep learning models. Thereafter, in the next section, we report the performance results of TensorFlow in training deep learning models. In the first experiment, we analyze the latency of TensorFlow in Sim mode and HW mode, and make a comparison with native versions using glibc and musl libc (i.e., running TensorFlow Lite with Ubuntu and Alpine linux) and a system provided by Intel using Graphene [23]. Graphene is an open-source SGX implementation of the original Graphene library OS. It follows a similar principle to Haven [33], by running a complete library OS inside of SGX enclaves. Similar to SCONE, Graphene offers developers the option to run their applications with Intel SGX without requiring code modifications. All evaluated systems except the Graphene-based system run inside a Docker container.

To have a fair comparison, the evaluated systems run with single thread because of the current version of the Graphene-based system does not support multiple threads, i.e., to run the classification process, we use the same input arguments for the classification command line:

Models. For classifying images, we use several pre-trained deep learning models with different sizes including Inception v3 [34] with the size of 91MB, Inception-v4 [35] with the size of 163MB and Densenet [36] with the size of 42MB. We manually checked the correctness of a single classification by classifying the image with the TensorFlow label_image application involving no self-written code and running directly on the host without containerization. We later compared the results to the ones provided by TensorFlow and other evaluated systems, we could confirm that indeed the same classifying result was produced by the evaluated systems.

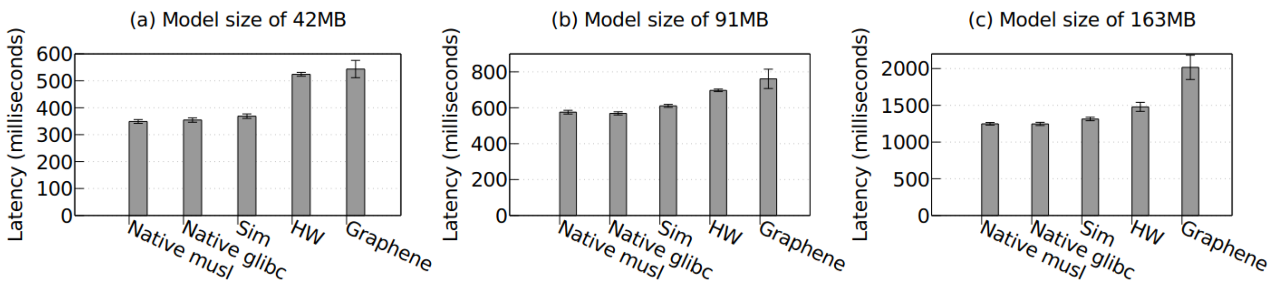


Figure 15: Comparison between TensorFlow, native versions and the state-of-the-art Graphene system in terms of latency with different model sizes.

6.1.5 Effect of input model sizes.

Figure 15 shows the latency comparison between TensorFlow with Sim andHW mode, native TensorFlow Lite with glibc, native TensorFlow Lite with musl libc, and Graphene-based system. TensorFlow with Sim mode incurs only 5% overhead compared to the native versions with different model sizes which is below the promised overhead concerning the KPIs (<30%). In addition, TensorFlow with Sim mode achieves a latency 1.39×, 1.14×, and 1.12× lower than TensorFlow with HW mode with the model size of 42MB, 91MB, and 162MB, respectively. This means that operations in the libc of TensorFlow introduce a lightweight overhead.

This is because TensorFlow handles certain system calls inside the enclave and does not need to exit to the kernel. In the Sim mode, the execution is not performed inside hardware SGX enclaves, but TensorFlow still handles some system calls in userspace, which can positively affect performance. We perform an analysis using strace tool to confirm that some of the most costly system calls of TensorFlow are indeed system calls that are handled internally by the SCONE runtime.

Interestingly, the native TensorFlow Lite running with glibc is the same or slightly faster compared to the version with musl libc. The reason for this is that both C libraries excel in different areas, but glibc has the edge over musl in most areas, according to microbenchmarks, because glibc is tailored for performance, whereas musl is geared towards small size. Because of this difference in goals, an application may be faster with musl or glibc, depending on the performance bottlenecks that limit the application. Differences in the performance of both C libraries must therefore be expected.

In comparison to Graphene-based-system, TensorFlow with HW mode is faster in general, and also faster than the Graphene-based-system when we increase the size of input models, especially when it exceeds the limit of the Intel SGX EPC size (94MB). In particular, with the model size of 42MB, TensorFlow with HW mode is only 1.03× faster compared to Graphene-based system, however, with the model size of 163MB, TensorFlow with HW mode is 1.4× compared to Graphene-based system.

The reason for this is that when the application allocates memory size larger than the EPC size limit, the performance of reads and writes is severely degraded because it performs encrypting data and paging operations which are very costly. To reduce this overhead, we reduce the size of our libraries loaded into SGX enclaves. Instead of adding the whole OS libc into SGX enclaves as Graphene did, we make use of SCONE libc variant which is a modification of musl libc having much smaller size. In SCONE, system calls are not executed directly but instead are forwarded to the outside of an enclave via the asynchronous system call interface.

This interface together with the user level scheduling allows TensorFlow to mask system call latency by switching to other application threads. Thus, we expect this speedup factor of TensorFlow compared to Graphene-based-system will increase more when the size of the input model size is increased and when the application runs with multiple threads.

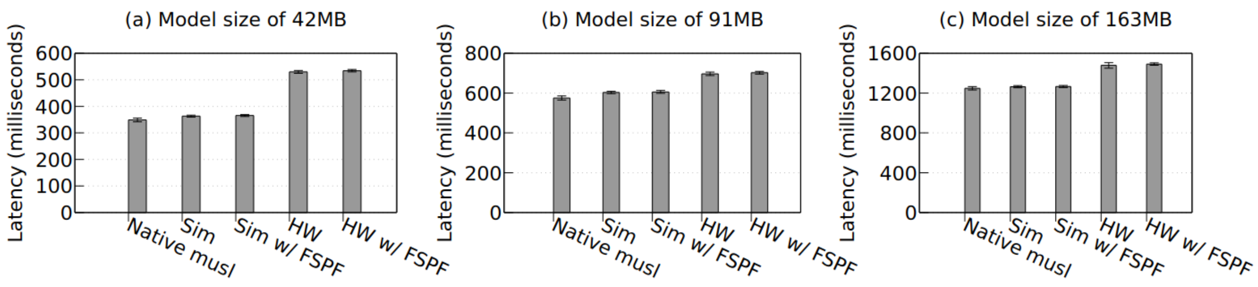


Figure 16: The effect of file system shield on the classification latency with different model sizes.

6.1.6 Effect of file system shield.

One of the real world use cases of TensorFlow is that a user not only wants to acquire classifying results but also wants to ensure the confidentiality of the input images since they may contain sensitive information, e.g., handwritten document images or concerning the federated learning use case, sensitive patient data. At the same time, the user wants to protect her/his machine learning models since he/she had to spend a lot of time and cost to train the models. To achieve this level of security, the user activates the file system shield of TensorFlow which allows he/she to encrypt the input including images and models and decrypt and process them within an SGX enclave.

In this experiment, we evaluate the effect of this file system shield on the overall performance of TensorFlow. As previous experiments, we use the same input Cifar-10 images. Figure 16 shows the latency of TensorFlow when running with/without activating the file system shield with different models.

The file system shield incurs significantly small overhead on the performance of the classification process. TensorFlow with Sim mode running with the file system shield is 0.12% slower than TensorFlow with Sim mode running without the file system shield. Whereas in the TensorFlow with HW mode, the overhead is 0.9%. The lightweight overhead comes from the fact that our file system shield uses Intel-CPU-specific hardware instructions to perform cryptographic operations and these instructions can reach a throughput of up to 4 GB/s, while the model is about 163 MB in size. This leads to a negligible overhead on the startup of the application only.

6.1.7 Continuous Runtime Benchmarks

The runtime security of SCONE is furthermore continuously evaluated using several standard benchmarks such as TPC-C benchmark and MariaDB) where its native performance is compared with when running in Intel SGX enclaves using SCONE. The average performance over the last three month revealed that we can achieve 4500 Tpmc vs. 6618.000 TpmC when running without SCONE.

6.2 Network Shield Evaluation

In this section, we present the results of both performance and functionality evaluation for our network shielding implementation. Real-world software service deployments often involve multiple stages of interconnected servers (e.g., application and database servers), whose data transfers must be protected.

We first present the results of latency and throughput performance benchmarks, using the well-known web servers nginx and Apache httpd as example applications. This analysis is relevant for AI applications whenever inference or federated learning are performed through TCP connections in distribution. After establishing a baseline by executing the servers both natively and using unshielded SCONE, we observe the impact of enabling the Network Shield for connections between them. For the experiments, we solely run in simulation mode as we only want to measure the impact of the shield itself. We show that using the AES-GCM implementation built into mbedtls incurs a heavy performance penalty, which can be largely mitigated by the help of the optimized replacement implementation. In particular, we compare the overhead of the network shield to the one of

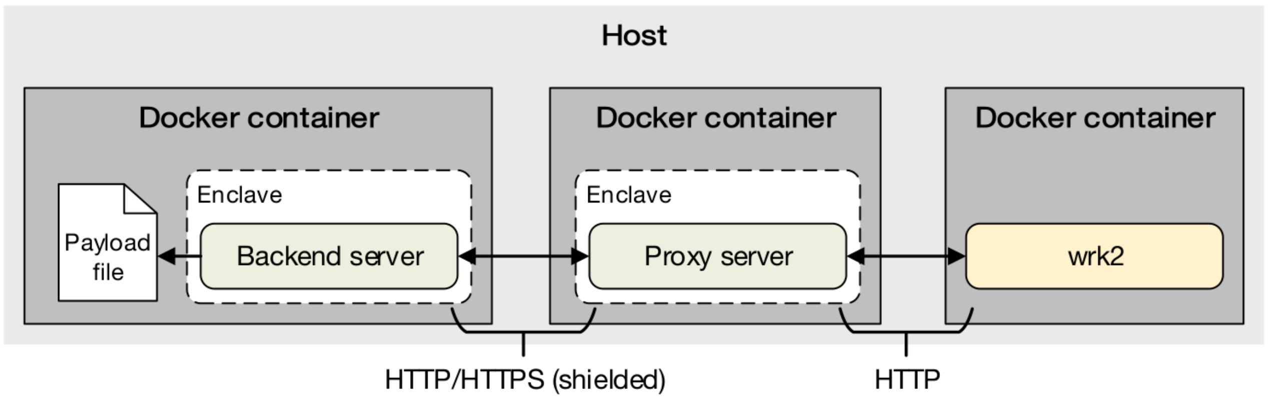


Figure 17: General evaluation benchmark setup.

application-provided OpenSSL-based TLS encryption, revealing that the Network Shield can achieve up to 76% of the latter's throughput in the case of nginx and 93% in the case of Apache httpd which is representative considering that collaborative learning application described in Section 2 involved data exchange through https channels.

Most of the time it is possible to use persistent connections between servers, which has been reflected by the prior setup. However, when interacting with external servers, this may not be feasible or desirable (in the case of infrequent use). Therefore, we also analyze the performance impact of handling non-persistent connections. The results show a decreased throughput when compared to persistent connections, which we can trace to the overhead introduced by repeated TLS handshakes. In this case, the network shield achieves only 17% to 24% of OpenSSL's throughput. We determine an inefficient implementation of asymmetric cryptography in mbedtls as the root cause and propose several options to reduce its performance impact. Additionally, we explore the influence of the I/O pattern changes caused by transparent TLS handshakes on resource usage. It is observed that, by default, they can cause a severe increase in CPU usage for applications employing asynchronous I/O event loops, such as nginx. We demonstrate that this can be averted by applying the asynchronous I/O event remapping mechanism which we are currently implementing.

6.2.1 Benchmark Setup

The setup of the benchmark is composed of a backend web server, a load generator acting as a client and a proxy web server located in between, as shown in Figure 5.4. This setup mimics real-world multi-stage deployments while simplifying the architecture to ease the experimentation. All components are started within Docker containers, which use the host network. The client always connects to the proxy server using HTTP. The connection between proxy and backend server uses HTTP, HTTPS with TLS being performed by the OpenSSL library configured through the web servers themselves or HTTP/HTTPS on top of the network shield using the mbedtls library, depending on the test case. We make use of wrk25 as a high-performance load generator and measurement tool. wrk2 produces a constant-throughput stream of requests and avoids coordinated omission effects through the use of HdrHistograms. As a result, we expect a spike in latency once the servers' saturation points are reached.

Comparisons to multi-host setups have shown that the network becomes the primary bottleneck. Consequently, we chose to conduct the experiments on a single host running all three components. We used a machine with a 6-core Intel Xeon E-2186G CPU @3.80 GHz and 32 GB of DRAM. The host runs Ubuntu 20.04 with 64-bit Linux kernel 5.4.0. We use Docker version 20.10.1 and containers based on Alpine Linux 3.7 and 3.8. To increase benchmark consistency, we disabled Hyperthreading, enabled the Linux performance CPU governor, pinned both server and client processes to individual CPU cores using CPU affinity masks and favoured their execution by using a nice value of 10 to increase the processes' scheduling priority. The web servers are operated with a minimal configuration. When using built-in HTTPS, the usage of the same cipher suites and elliptic curves as designated by the Network Shield (ECDHE-ECDSA-AES256-GCM-SHA384 and secp384r1) has been enforced and a similar certificate hierarchy has been deployed that contains 5 certificates, deviating only in key pairs and certificate common names. These certificate differences are unavoidable since SCONE benchmarks use regular CAS operations to provision freshly generated keys and certificates. Note, however, that no client authentication was performed when utilizing built-in HTTPS, as the web server implementations were unable to authenticate clients sending a multi-level client certificate hierarchy successfully. The network shield, on the other hand, performs client authentication.

SCONE will be enabled for the backend and proxy servers, wrk2 on the other hand will always be executed natively. SCONE will be executed in simulation mode (sim) which does not use Intel SGX enclaves, it rather performs all of SCONE's operations in a normal process environment. SCONE configuration is provisioned by a CAS instance allocated for each benchmark run, backend and proxy server configuration reside within the same CAS session. Different session configurations will be used, depending on the type of benchmark.

6.2.2 nginx Throughput/Latency Benchmark

We use Docker containers based on a single-threaded nginx 1.14.2 as instances of the backend and proxy server introduced in the general setup. poll is used as the asynchronous event loop method; other methods have shown to yield similar results.

wrk2 is configured to use 1 thread and 10 connections, which are sufficient to fully saturate the servers. All connections are marked as keep-alive connections, i.e., the following measurements do not show any connection establishment overhead, as the same connections will be re-used for the entirety of the experiments. We perform benchmarks by continuously issuing requests to retrieve a static payload, hosted at the backend server, over the proxy server to the wrk2 client, increasing the constant-throughput rate every 60 seconds. We record the system's behavior for two different payloads. We observed an inconsistent achievable maximum throughput between backend and proxy server restarts for all configurations (including native execution). In order to present consistent results, we repeatedly probed different server instances and subsequently chose one which yielded results in the top 20automated fashion.

6.2.3 Baseline

The first benchmark uses a 64 KiB randomly generated binary file as its static payload. We measure the latency/throughput while increasing the induced request rate for each recorded data point. Figure 5.5 shows the experiment's baseline, using either HTTP or HTTPS (with TLS performed by OpenSSL) connections between the backend and proxy servers, when executing the servers natively (i.e., without SCONE) and with SCONE (unshielded, i.e., without activating the Network Shield). For native HTTP and HTTPS we observe a mostly stable latency, averaging 1.25ms and 1.24ms per request respectively, until the servers' saturation points are reached, which happens at 33,400req/s for HTTP and 17,500req/s for HTTPS, at which point the latency spikes. We defined the cutoff to be one second. Enabling SCONE naturally entails an overhead as system calls will be filtered, processed and exchanged between enclave and kernel using dedicated system call threads (s-threads). This overhead is more severe in hardware mode, where Intel SGX enclave memory limitations apply. Having SCONE enabled in hardware mode worsens the achievable maximum throughput rate to 20,900req/s for HTTP and 12,300req/s for HTTPS. The latency is slightly higher, at an average of 1.75ms (HTTP) and 1.88ms (HTTPS) considering throughput rates between 2500req/s and their respective saturation points. At lower rates, notably 500req/s, 1000req/s and 2000 req/s, the latency experienced with SCONE servers is higher than the average – up to 4.1ms. This is most likely caused by the behavior of s- threads, which tend to enter longer sleep periods upon low rates, although the exception of the regular low latency at a rate of 1500req/s remains unclear. As the throughput rate increases towards its maximum, the latency of SCONE configurations gradually decreases, getting closer to native latency.

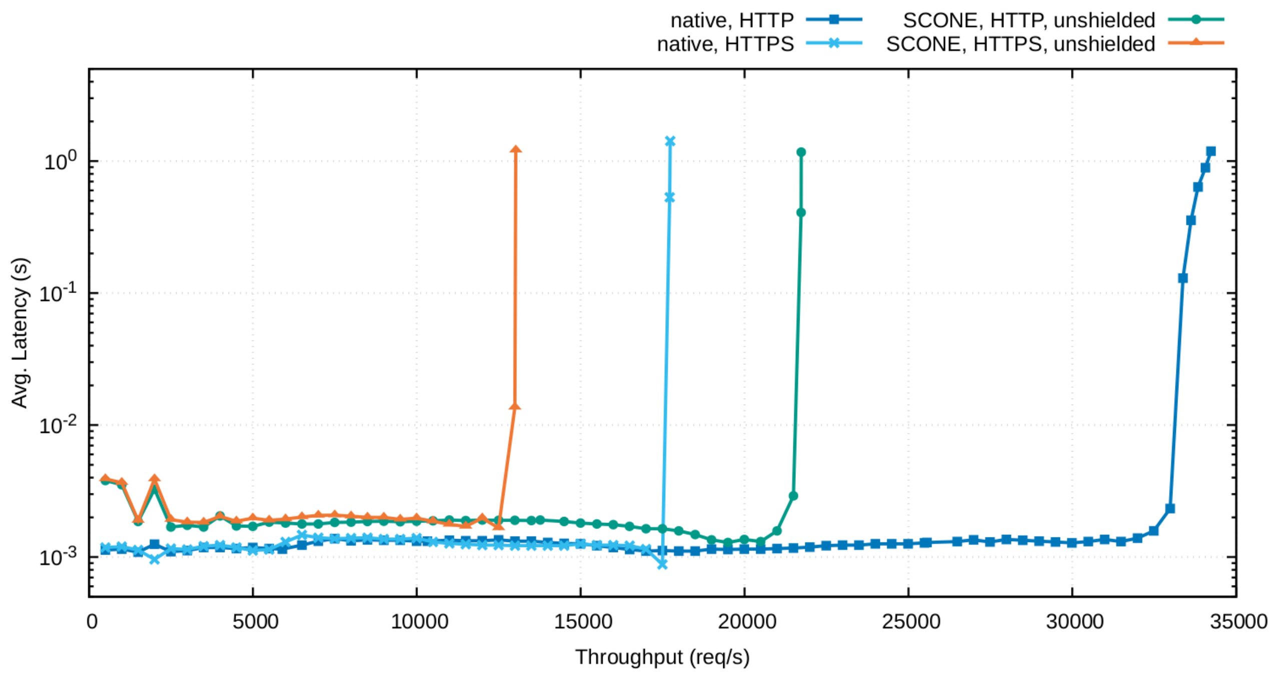


Figure 18: Network Shield latency/throughput development for native and unshielded SCONE HTTP(S) configurations.

6.2.4 Network Shield Overhead

The network shield features different modes of operation – we evaluate both unprotected mode (which does not feature any filtering or encryption, i.e., should be equal to unshielded execution) and protected mode (which enables mbedtls-based TLS encryption, i.e., should show some overhead). Figure 19 shows the benchmark results for both modes. We also observe the limitations of the AES-GCM implementation built into mbedtls and compare its performance to the optimized alternative we investigated.

As expected, unprotected mode yields the same performance, both latency- and throughput-wise, as the unshielded configuration seen in the previous graph. Enabling the shield's protected mode using the AES- GCM implementation built into mbedtls shows a substantial performance loss: Only 1500req/s remain of the previously unshielded 20,900req/s, reducing throughput by almost 93% . Using the optimized Intel AES-GCM implementation, on the other hand, the network shield's protected mode is able to achieve up to 9350req/s. While this still exhibits a higher overhead than unshielded HTTPS (using OpenSSL) at 76% of the latter's maximum rate, it shows a considerable improvement over the implementation provided by mbedtls by default. At an average of 2.15ms, the shield's latency is also higher than the 1.88ms exhibited by the OpenSSL-using variant.

To show that the throughput discrepancy is still largely dominated by the implementation of mbedtls(including the replacement AES-GCM implementation), we repeat the experiment with a modified version of the network shield, which features exactly the same code paths as the prior protected mode, but replaces the calls to the mbedtls library (mbedtls_ssl_read and mbedtls_ssl_write) with their underlying Basic Input/Output (BIO) callbacks (network_shield_mbedtls_unshielded_recv and network_shield_mbedtls_unshielded_send), skipping the encryption and decryption provided by mbedtls.

This is shown as the "protected/no TLS" curve in the same graph. Doing so yields a maximum throughput rate of 20,200req/s, which is 96.7% of the same value for unshielded HTTP, at similar latencies. The remaining overhead, imposed by the network shield's system call filtering and re-writing, is therefore much lower than the discrepancy between unshielded HTTPS (utilizing OpenSSL integrated into nginx) and protected HTTP (utilizing mbedtls).

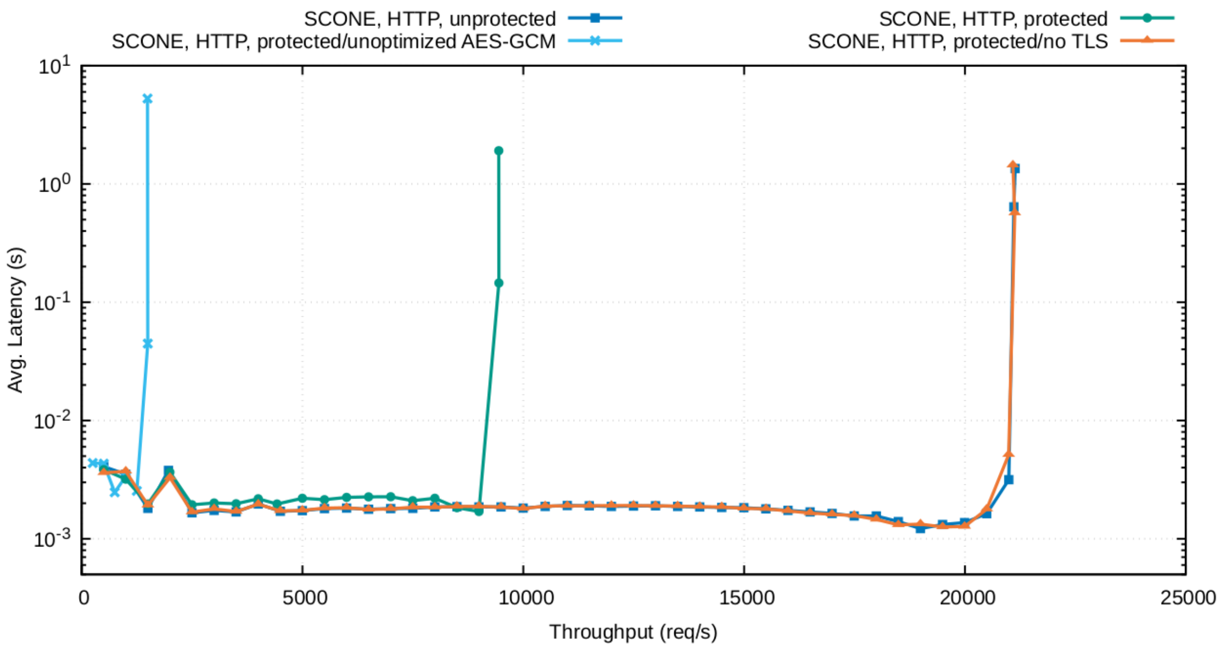


Figure 19: Network Shield latency/throughput development for unprotected and protected environment.

7 Summary & Conclusions

This document presents the first iteration and release of the Data Broker we developed within the NEARDATA platform. It covers the effort of the initial twelve months which involves the formulation of policies to address various security threats and attacks targeting Extreme Data applications. The document outlines the mechanisms and security tools we have created to improve the security of applications running within NEARDATA. In order to run NEARDATA applications in Trusted Execution Environments, we utilized and extended the following technologies: The SCONE runtime, cross compiler, sconify image tool, and the Configuration and Attestation Service (CAS). Each of these components are described in detail, including its specific functionality and its location within the provided use cases such as the federated learning and the Data Hub Metabolomics application. Additionally, the document highlights the efforts made to ensure secure network communications, covering edge communication and communication within cloud environments through transparent encryption provided by the SCONE network shield.

With the release of this deliverable, we have achieved the third objective of NEARDATA, which is to establish a Data Broker service that enables secure and reliable exchange of data and coordination of data pipelines across the Compute Continuum. To ensure the security and reliability of the data, we have furthermore extended applications such as federated learning to run in Trusted Execution Environments (TEEs). We have also implemented methods for transparent encryption to ensure the security of data during transmission and storage. These mechanisms are designed to ensure high throughput and do not require any modifications to the existing code. We also integrated Keycloak (an open-source identity and access manager) into Lithops to manage user authentication and policies. This serves as Identity Provider within the Data Broker's architecture.

A summary of the KPIs for Data Broker components is illustrated in Table 13

We are currently working on integrating SCONE and Lithops to showcase the Data Broker (CAS) functionality for future development. In the future, we plan to expand on SCONTAIN's efforts in developing a fault-tolerant TPM-based CAS. During the architectural design phase, we will incorporate Intel TDX, AMD SEV, and ARM CCA at the virtual machine (VM) level instead of the process

Use-case	**KPI**	**Results**
Surgery	KPI-4	We achieved the Key Performance Indicator (KPI) by the following measures we have taken: • Examine how security such as confidentiality and integrity can be provided for applications such as federated learning by harnessing technologies such as Trusted Execution Environments (TEEs). • Assessing the performance degradation of the confidential federated learning application by analysing the impact of the network shield, configured through the Data Broker's CAS component, on execution time. Comparing the impact to both, the Vanilla (Intel/Non-SGX) environment and the SCONE (Intel SGX/hardware mode) setup, with a particular focuses on the execution time of the federated learning use case. • Assessing the impact on the initial start up and response time of functions in the federated learning application frameworks such as Flower.
Metabolomics	KPI-4	We have integrated Confidential Compute Layer component of the Data Broker into Lithops and tested its early usage on an on-premise K8s cluster

Table 13: Highlights of main KPIs achieved by the Data Broker.

level. Additionally, we will enhance the network shield to attest also the communication partners.

References

[1] J. Konečný, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," arXiv preprint arXiv:1610.05492, 2016.

[2] P. Voigt and A. Von dem Bussche, "The eu general data protection regulation (gdpr)," A Practical Guide, 1st Ed., Cham: Springer International Publishing, vol. 10, no. 3152676, pp. 10–5555, 2017.

[3] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang, "Deep learning with differential privacy," in Proceedings of the 2016 ACM SIGSAC conference on computer and communications security, pp. 308–318, 2016.

[4] V. Mugunthan, A. Peraire-Bueno, and L. Kagal, "Privacyfl: A simulator for privacy-preserving and secure federated learning," in Proceedings of the 29th ACM International Conference on Information & Knowledge Management, pp. 3085–3092, 2020.

[5] M. Fang, X. Cao, J. Jia, and N. Gong, "Local model poisoning attacks to {Byzantine-Robust} federated learning," in 29th USENIX security symposium (USENIX Security 20), pp. 1605–1622, 2020.

[6] C. Xie, O. Koyejo, and I. Gupta, "Fall of empires: Breaking byzantine-tolerant sgd by inner product manipulation," in Uncertainty in Artificial Intelligence, pp. 261–270, PMLR, 2020.

[7] R. Kunkel, D. L. Quoc, F. Gregor, S. Arnautov, P. Bhatotia, and C. Fetzer, "Tensorscone: A secure tensorflow framework using intel sgx," arXiv preprint arXiv:1902.04413, 2019.

[8] D. L. Quoc, F. Gregor, S. Arnautov, R. Kunkel, P. Bhatotia, and C. Fetzer, "Securetf: A secure tensorflow framework," in Proceedings of the 21st International Middleware Conference, pp. 44–59, 2020.

[9] A. N. Bhagoji, S. Chakraborty, P. Mittal, and S. Calo, "Analyzing federated learning through an adversarial lens," in International Conference on Machine Learning, pp. 634–643, PMLR, 2019.

[10] P. Blanchard, E. M. El Mhamdi, R. Guerraoui, and J. Stainer, "Machine learning with adversaries: Byzantine tolerant gradient descent," Advances in neural information processing systems, vol. 30, 2017.

[11] V. Costan and S. Devadas, "Intel sgx explained," IACR Cryptol. ePrint Arch., vol. 2016, p. 86, 2016.

[12] X. Cao, J. Jia, and N. Z. Gong, "Provably secure federated learning against malicious clients," in Proceedings of the AAAI conference on artificial intelligence, vol. 35, pp. 6885–6893, 2021.

[13] F. Gregor, W. Ozga, S. Vaucher, R. Pires, S. Arnautov, A. Martin, V. Schiavoni, P. Felber, C. Fetzer, et al., "Trust management as a service: Enabling trusted execution in the face of byzantine stakeholders," in 2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), pp. 502–514, IEEE, 2020.

[14] P. Karnati, "Data-in-use protection on ibm cloud using intel sgx," IBM, May, 2018.

[15] 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.

[16] M. Bailleu, D. Giantsidi, V. Gavrielatos, V. Nagarajan, P. Bhatotia, et al., "Avocado: A secure {In-Memory} distributed storage system," in 2021 USENIX Annual Technical Conference (USENIX ATC 21), pp. 65–79, 2021.

[17] R. Krahn, D. Dragoti, F. Gregor, D. L. Quoc, V. Schiavoni, P. Felber, C. Souza, A. Brito, and C. Fetzer, “Teemon: A continuous performance monitoring framework for tees,” in Proceedings of the 21st International Middleware Conference, pp. 178–192, 2020.

[18] D. Le Quoc, F. Gregor, J. Singh, and C. Fetzer, “Sgx-pyspark: Secure distributed data analytics,” in The World Wide Web Conference, pp. 3564–3563, 2019.

[19] W. Ozga, D. L. Quoc, and C. Fetzer, “Perun: Confidential multi-stakeholder machine learning framework with hardware acceleration support,” in IFIP Annual Conference on Data and Applications Security and Privacy, pp. 189–208, Springer, 2021.

[20] W. Ozga, D. L. Quoc, and C. Fetzer, “A practical approach for updating an integrity-enforced operating system,” in Proceedings of the 21st International Middleware Conference, pp. 311–325, 2020.

[21] W. Ozga, D. L. Quoc, and C. Fetzer, “Weles: Policy-driven runtime integrity enforcement of virtual machines,” arXiv preprint arXiv:2104.14862, 2021.

[22] J. Singh, J. Cobbe, D. L. Quoc, and Z. Tarkhani, “Enclaves in the clouds: Legal considerations and broader implications,” Communications of the ACM, vol. 64, no. 5, pp. 42–51, 2021.

[23] C.-C. Tsai, D. E. Porter, and M. Vij, “{Graphene-SGX}: A practical library {OS} for unmodified applications on {SGX},” in 2017 USENIX Annual Technical Conference (USENIX ATC 17), pp. 645–658, 2017.

[24] V. Costan, I. Lebedev, and S. Devadas, “Sanctum: Minimal hardware extensions for strong software isolation,” in 25th USENIX Security Symposium (USENIX Security 16), pp. 857–874, 2016.

[25] A. Martin, C. Lian, F. Gregor, R. Krahn, V. Schiavoni, P. Felber, and C. Fetzer, “Adam-cs: Advanced asynchronous monotonic counter service,” in 2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), pp. 426–437, IEEE, 2021.

[26] V. Scarlata, S. Johnson, J. Beaney, and P. Zmijewski, “Supporting third party attestation for intel® sgx with intel® data center attestation primitives,” White paper, p. 12, 2018.

[27] D. J. Beutel, T. Topal, A. Mathur, X. Qiu, J. Fernandez-Marques, Y. Gao, L. Sani, K. H. Li, T. Parcollet, P. P. B. de Gusmão, et al., “Flower: A friendly federated learning research framework,” arXiv preprint arXiv:2007.14390, 2020.

[28] G. D. P. Regulation, “General data protection regulation (gdpr),” Intersoft Consulting, Accessed in October, vol. 24, no. 1, 2018.

[29] A. Krizhevsky, G. Hinton, et al., “Learning multiple layers of features from tiny images,” 2009.

[30] Y. LeCun, C. Cortes, C. Burges, et al., “Mnist handwritten digit database,” 2010.

[31] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 770–778, 2016.

[32] B. Xu, N. Wang, T. Chen, and M. Li, “Empirical evaluation of rectified activations in convolutional network,” arXiv preprint arXiv:1505.00853, 2015.

[33] A. Baumann, M. Peinado, and G. Hunt, “Shielding applications from an untrusted cloud with haven,” ACM Transactions on Computer Systems (TOCS), vol. 33, no. 3, pp. 1–26, 2015.

[34] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, “Rethinking the inception architecture for computer vision,” in Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 2818–2826, 2016.

[35] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. Alemi, “Inception-v4, inception-resnet and the impact of residual connections on learning,” in Proceedings of the AAAI conference on artificial intelligence, vol. 31, 2017.

[36] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, “Densely connected convolutional networks,” in Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 4700–4708, 2017.