



Project Title	Artificial Intelligence in Secure PRIVacy-preserving computing coNTinuum
Project Acronym	AI-SPRINT
Project Number	101016577
Type of project	RIA - Research and Innovation action
Topics	ICT-40-2020 - Cloud Computing: towards a smart cloud computing continuum (RIA)
Starting date of Project	01 January 2021
Duration of the project	36 months
Website	www.ai-sprint-project.eu/

D4.1 - Initial Release and Evaluation of the Security Tools

Work Package	WP4 Security Tools
Task	T4.1 Data Security, T4.2 Application Execution Security, T4.3 Network Security and T4.4 Patch Management and Secure Boot
Lead author	André Martin (TUD), Giacomo Verticale (POLIMI)
Contributors	André Martin (TUD), Mahshid Mehrabi (TUD), Giacomo Verticale (POLIMI), Davide Testoni (POLIMI), Elia Battiston (POLIMI)
Peer reviewers	Danilo Ardagna (POLIMI), Francesc Lordan (BSC)
Version	V1.6
Due Date	31/12/2021
Submission Date	24/12/2021

Dissemination Level

<input checked="" type="checkbox"/>	PU: Public
<input type="checkbox"/>	CO: Confidential, only for members of the consortium (including the Commission)
<input type="checkbox"/>	EU-RES. Classified Information: RESTREINT UE (Commission Decision 2005/444/EC)
<input type="checkbox"/>	EU-CON. Classified Information: CONFIDENTIEL UE (Commission Decision 2005/444/EC)
<input type="checkbox"/>	EU-SEC. Classified Information: SECRET UE (Commission Decision 2005/444/EC)



AI-Sprint - Artificial Intelligence in Secure PRIVacy-preserving computing coNTinuum, has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement no. 101016577.

Versioning History

Revision	Date	Editors	Comments
0.1	10/11/2021	André Martin, Mahsid Mehrabi	ToC definition
0.2	17/11/2021	André Martin	Updated ToC based on feedback
0.3	26/11/2021	André Martin	Added content for federated learning
0.4	29/11/2021	André Martin	Added content for SCONE/Runtime security
0.5	01/12/2021	André Martin	Added content for SCONE CAS, network part
0.6	02/12/2021	André Martin	Added content for performance evaluation
0.7	03/12/2021	Giacomo Verticale	Added content for edge network part
0.8	06/12/2021	André Martin	Complete pass over text - homogenization
0.9	07/12/2021	André Martin	Added last missing content items such as sconify image, fixed figure labels and renumbering
1.0	10/12/2021	Giacomo Verticale	Added references to the network security mechanisms. Fixed captions
1.1	13/12/2021	Danilo Ardagna	Review of the document
1.2	14/12/2021	Francesc Lordan Gomis	Review of the document
1.3	15/12/2021	André Martin	Incorporating review feedback
1.4	16/12/2021	André Martin	Incorporating review feedback and adding architectural figure
1.5	17/12/2021	André Martin	Adding performance measurements for runtime security and finalizing
1.6	20/12/2021	Giacomo Verticale	Incorporating review feedback

Glossary of terms

Item	Description
AI	Artificial Intelligence
CAS	Configuration and Attestation Service
CRD	Custom Resource Definitions
CRR	Custom Resource Records
CVE	Common Vulnerabilities and Exposures
ECDHE	Elliptic--curve Diffie-Hellman Ephemeral
EWMA	Exponential Weighted Moving Average
GPU	Graphics Processing Unit
MEC	Mobile Edge Computing
MPLS	Multi Protocol Label Switching
OS	Operating System
RAN	Radio Access Network
SR-MPLS	Segment Routing for MPLS
TEE	Trusted Execution Environment
TLS	Transport Layer Security
UPF	User Plane Function

Keywords

Artificial Intelligence; Edge Computing; Computing Continuum; Security; Runtime management.

Disclaimer

This document contains confidential information in the form of the AI-SPRINT project findings, work and products and its use is strictly regulated by the AI-SPRINT Consortium Agreement and by Contract no. 101016577.

Neither the AI-SPRINT Consortium nor any of its officers, employees or agents shall be responsible, liable in negligence, or otherwise however in respect of any inaccuracy or omission herein.

The contents of this document are the sole responsibility of the AI-SPRINT consortium and can in no way be taken to reflect the views of the European Commission and the REA.

Executive Summary

This document describes the first release and evaluation of the security tools developed by the AI-SPRINT project, while the second release and evaluation of the security tools are due at M24.

The focus of this document is to describe the components involved in this first release, which harden the continuous deployment and programming framework runtime developed in WP3 with regards to several security related aspects, as well as the results of the preliminary tests on the technologies employed that support the design decisions.

In this document, we first describe an example application, i.e., federated learning, which we will reference throughout this document in order to describe the different security tools we developed within AI-SPRINT and how they ensure confidentiality as well as integrity in this context.

Although federated learning provides a certain degree of confidentiality by default, as each party is performing the learning only locally on premise and sharing only model parameters with its other collaborators, there are still many aspects that require tooling around existing federated learning frameworks in order to guarantee the privacy constraints we put forth in the AI-SPRINT project.

After the introduction of the federated learning example application, we review the threat model we consider in AI-SPRINT as well as possible attacks and will refer to them later in the policy definitions that application developers and users will use to define their requirements.

Next, the document describes systematically the different protection goals stemming from the previous analysis of threats typically found in the context of AI applications running in cloud environments as well as on edge devices. For the policy definitions, which are based on the previously defined protection goals, a policy language using YAML file syntax is presented.

The policy language can be used in two ways, i.e., developers and application users can either define the requirements with respect to security either by stating their protection goals or by explicitly selecting the protection mechanism that should be enabled when the application is being deployed and run on the distributed infrastructure.

In addition to the previously declared policy description representing the protection goals as well as protection mechanisms, we then review how security measures can be fine-tuned using the so-called SCONE session language.

After the introduction of the policy language, the document describes the different security tools we implemented within the scope of AI-SPRINT starting from the overall objectives in order to provide confidential compute following with a thorough description of components that fulfil these objectives.

The tools we present range from the so-called SCONE runtime, the cross compiler as well as the sconify tool that allows developers to turn native applications into confidential ones. We also present the CHIMA framework for deploying network functions, in particular security functions, to programmable switches and a blockchain-based mechanism for authorizing access to data collected from devices in the field.

Finally, a few micro benchmarks are being presented evaluating the performance of various aspects of the system. The deliverable concludes with a summary of achievements.

Table of Contents

1. Introduction	7
1.1 Scope of the document	7
1.2 Target Audience	7
1.3 Structure of document	7
2. Example Application: Federated Learning	8
3. AI-SPRINT Security Overview	11
3.1. Threat Model	11
3.2. Attacks In The Context of AI-SPRINT	11
3.3. Security Policies Introduction and Overview	13
3.4. Policy Definitions	13
3.5. Fine Grained Policy Definitions	15
3.6. Policy Server and Architecture	17
4. First Release of the Security Tools	20
4.1. Security Tools	21
4.2. Network Security	35
5. Performance Evaluation	48
5.1. Runtime Security	48
5.2. Network Shield Evaluation	51
5.3. Evaluation of the CHIMA Framework	55
5.4. Evaluation of the Blockchain Authentication Mechanism	58
6. Towards the Integrated framework for the Security Tools	60
6.1. Integration plan for the Security Tools	60
7. Summary & Conclusions	62
References	63

List of Tables

Table 3.1 - Attacks and security mechanisms to mitigate them	13
Table 3.2 - Protection Goals / Options	14
Table 4.1 - Intel SGX SDK vs. SCONE	25
Table 5.1 - Breakout of redeployment times for different topologies. 95% CI	60
Table 5.2 - Average time to fetch a smart contract from the blockchain	61

List of Figures

Figure 2.1 - Federated Learning Architecture	9
Figure 3.1 - Security tools architecture	18
Figure 3.2 - Security tools architecture - left CLI tools, middle runtime with TEE Hardware support, right runtime if no TEE hardware support is available.	19
Figure 3.3 - 5G Industrial network comprising a private slice and a public slice, a private MEC, a public MEC, and a public cloud. The AI-SPRINT Security Gateway is placed between the UPF and the MEC or cloud.	20
Figure 4.1 - Program arguments as well as environment variables are provided through CAS	28
Figure 4.2 - Example who one entity is being authenticated using TLS	28
Figure 4.3 - Example about how two entities authenticate each other using TLS - mutual authentication	29
Figure 4.4 - Example of transparent network encryption. Connections are terminated inside the enclaves	29
Figure 4.5 - SCONE file system shield - left volume that is encrypted, an integrity protected but not encrypted volume (middle), right no protection at all	30
Figure 4.6 - Control flow of transparent encryption system operations	38
Figure 4.7 - Architectural domains of the blockchain-based authentication, authorization and access control mechanism	42
Figure 4.8 - User authentication	44
Figure 4.9 - Authorization of a user query	45
Figure 4.10 - An heterogeneous service chain addressing both P4 programmable switches and Docker containers.	47
Figure 4.11 - The components of the CHIMA framework.	48
Figure 4.12 - Usage of SR-MPLS in CHIMA.	49
Figure 5.1 - The attestation and keys transferring latency comparison between TensorFlow with the traditional way using IAS.	51
Figure 5.2 - Comparison between TensorFlow, native versions and the state-of-the-art Graphene system in terms of latency with different model sizes, (a) Densenet (42MB), (b) Inception_v3 (91MB), and (c) Inception_v4 (163MB).	52
Figure 5.3 - The effect of file system shield on the classification latency with different model sizes, (a) Densenet (42MB), (b) Inception_v3 (91MB), and (c) Inception_v4 (163MB).	53
Figure 5.4 - General evaluation benchmark setup	54
Figure 5.5 - Network Shield latency/throughput development for native and unshielded SCONE HTTP(S) configurations: nginx, 64 KiB payload, hw mode.	56
Figure 5.6 - Network Shield latency/throughput development for unprotected, protected and TLS-over-TLS unwrapped HTTPS configurations: nginx, 64 KiB payload, hw mode.	57
Figure 5.7 - Topologies used to evaluate the redeployment times.	
Figure 5.8 - Delay in the detection of an exceeded requirement with different intervals for the polling of new measurements from the INT collector. The bars indicate the 95% confidence interval	58
Figure 5.9 - Delay in the detection of an exceeded requirement with different coefficients for the computation of the EWMA on link measurements. The bars indicate 95% confidence intervals.	59
Figure 6.1 - Milestones for components development	63

1. Introduction

1.1 Scope of the document

The aim of the AI-SPRINT “Artificial intelligence in Secure PRiVacy-preserving computing coNTinuum” project is to develop a platform composed of design and runtime management tools to seamlessly design, partition and operate Artificial Intelligence (AI) applications among the current plethora of cloud-based solutions and AI-based sensor devices (i.e., devices with intelligence and data processing capabilities), providing resource efficiency, performance, data privacy, and security guarantees. This document describes the various security tools that ensure confidentiality as well as integrity for services running on the AI-SPRINT platform.

1.2 Target Audience

The release and evaluation of the security tools document (first and second) are intended for internal use, although it is publicly available. The target audience is the AI-SPRINT technical team including all partners involved in the delivery of work packages 2, 3 and 4 but also it serves as reference for the developers of the three use cases of the project.

1.3 Structure of document

This document includes three main parts:

- An introduction into the **federated example application**, which will be used throughout the document as a running example in order to demonstrate the various security related features we are developing within AI-SPRINT.
- The **AI-SPRINT Security Overview** provides a summary about potential threats and attacks in the context of 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.
- The **First Release of the Security Tools** 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 in runtime security as well as network security addressing different layers of the AI-SPRINT architecture.
- In the Conclusion section, we summarize our contributions and outline the ongoing and future work.

2. Example Application: Federated Learning

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 AI-SPRINT. As an example, we chose Federated Learning as it demonstrates 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 AI-SPRINT.

Federated Learning [12] 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 the way in which personal data is handled and processed such as EU's General Data Protection Regulation (GDPR) [28].

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 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 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 [1]. 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 [18].
3. Malicious clients can tamper their local training data or parameters updates forwarded to the central training system to corrupt the global model [8, 29].
4. The attacker compromises the sensor on the field and provides tampered data to the client.

To handle the issues (1) and (2), state-of-the-art solutions rely on a privacy-preserving mechanism such as differential privacy or secure multiparty computation (MPC). 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 [14, 15]. To cope with issue (3), several Byzantine-robust federated learning mechanisms have been proposed [5, 6, 8, 29]. 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 [5, 8, 29] 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 robust mechanism by carefully tampering with their model parameters sent to the central training system [7].

In AI-SPRINT, 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 [9, 11, Signh2021] as well as in academia [3, 4, 5, 13, 15, 16, 19, 20, 21, 22, 25, 27]. To ensure confidentiality and integrity of applications, TEEs execute their code and data inside an encrypted memory region called **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** [Costan2016] 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 [17].

We tackle issues (2) and (3) by developing a Security Policy Manager component called CAS (Configuration and Attestation Service) based on the remote attestation mechanism supported by TEEs [10, 24]. 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. To handle issue (4) we provide the blockchain-based authentication and authorization mechanism, which decouples the local client from the device in the field, allowing access to the device only to authorized entities and reducing the risk of an attacker compromising the sensor.

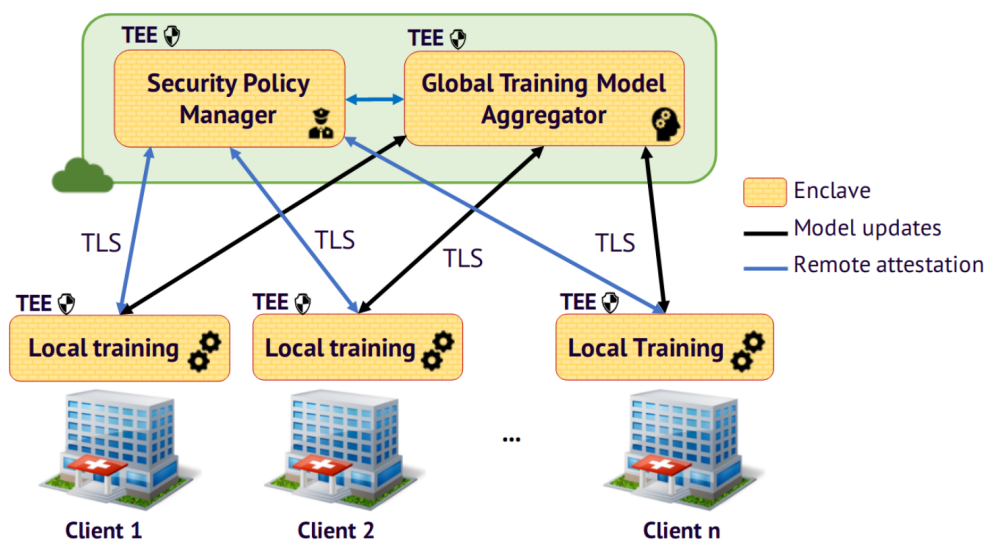


Figure 2.1 - Federated Learning Architecture

Figure 2.1 shows the architecture of the federated learning application in AI-SPRINT. The main goal of our hardened version is not only to ensure the confidentiality, integrity and freshness of input data, code, and machine learning models but also to enable multiple clients (who do not necessarily trust each other) to get the benefits of collaborative training without revealing their local training data. In the confidential setup, each client performs the local training also inside TEE enclaves to make sure that no one tampers with the input data or training code during the computations. To govern and coordinate the collaborative machine learning training computation among clients, we design a trusted management component, called Configuration and Attestation Service 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 Configuration and Attestation Service automatically and transparently performs remote attestation to make sure the local computations are running the correct code, correct input data, and on correct platforms as 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 Configuration and Attestation Service 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 [10][26].

After receiving the agreed security policies from clients, the Configuration and Attestation Service 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, 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 that is permitted to access the secrets reads the file.

We design the Configuration and Attestation Service in a way that we can delegate the 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 implemented the federated learning prototype using Intel OpenFL [23] - a distributed federated machine learning framework. We ran the local and global training computations inside SGX enclaves using SCONE [Arnatov2016] - a shielded execution framework to enable unmodified applications to run inside SGX enclaves. In the SCONE platform, the source code of an application is recompiled against a modified standard C library (SCONE libc) to facilitate the execution of system calls. The address space of the application stays within an enclave. In our prototype, the input training data and code are encrypted using the file system shield of SCONE, and then decrypted and processed inside SGX enclaves which cannot be accessed even by strong attackers with root access. We rely on our previous works [10, 28] to implement the Configuration and Attestation Service.

3. AI-SPRINT Security Overview

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

3.1. Threat Model

Edge Computing Systems and Mobile Edge Computing Systems (MEC) systems comprise various resources, which are shared among different applications and – in the case of public MECs – users. Therefore, it is possible that a compromised device spreads malicious traffic to fingerprint or access resources of other users. Given that Edge Computing is a relatively new technology, security mechanisms are less established and understood. In addition, in multi-tenant MECs, interactions can cause conflicts in network configurations [2].

For the threat model, 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. However, we assume that a malicious privileged software may be installed by an attacker with administrator rights.

In addition, 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. 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.

3.2. Attacks In The Context of AI-SPRINT

Considering the above listed threats, a malicious user can drive the following attacks in the context of AI applications: In order 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 exploiting their low complexity and lack of support for fine-grained access control policies. In AI-SPRINT, we tackle this attack by means of a Data Provider node, which intermediates any exchange between the devices and the users. The Data Provider uses a blockchain-based mechanism to manage fine-grained access policies.

Besides gaining access to confidential data, another attack vector is the maliciously introduction of wrong information in order to **tamper training results**. This can be achieved by a malicious user by 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 in order to pass the mutual authentication when establishing 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 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 attacks 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 anymore. 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.

We also consider the scenario of **lateral movement**, in which an attacker, after compromising a legitimate node, tries to access additional resources. AI-SPRINT tackles this attack by using the ability of 5G networks to authenticate devices and using micro-segmentation.

3.3. Security Policies Introduction and Overview

Security policies in AI-SPRINT define what level of protection the application user and developer desires for their application in order to mitigate the previously described attacks.

In order to define those policies, we will now list the several attacks and what mechanism mitigates them:

Attack	Mechanism
<ul style="list-style-type: none"> 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 	Trusted Execution Environment (TEE)
<ul style="list-style-type: none"> Rollback to old data (rollback protection for data at rest) 	Monotonic Counter (MC)
<ul style="list-style-type: none"> Reading & modifying input data for training as well as training models (confidentiality protection & integrity protection for data at rest) 	File system protection shield (FSPS)
<ul style="list-style-type: none"> Reading data exchanged over network during training (confidentiality protection & integrity protection for data in transit) 	Network protection shield (NPS)
<ul style="list-style-type: none"> Reading & modifying or forging input data for training as well as training models (confidentiality protection & integrity protection, lateral movement, tampering with training results for data at rest) 	5G device authentication
<ul style="list-style-type: none"> Reading input data for training as well as training models (confidentiality protection for data at rest). Reading input data for predictions (confidentiality protection for data in transit) 	Blockchain-based authentication and authorization

Table 3.1 - Attacks and security mechanisms to mitigate them.

3.4. Policy Definitions

In AI-SPRINT, application developers and users can describe their constraints with regards 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. In the first case, the policy server will infer what mechanisms need to be enabled in order to achieve the desired protection goal expressed by developers as code annotations (see AI-SPRINT deliverable *D2.1 First release and evaluation of the AI-SPRINT design tools*, Section 4.4) while, in the second case, the protection mechanisms are enabled as chosen by the application developer or user.

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 AI-SPRINT:

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)
Rollback protection rollback	Monotonic Counter (MC)
Authentication of 5G devices authentication	5G device authentication
Data access policy dataAccess	Blockchain-based authentication and data access control

Table 3.2 - Protection Goals / Options

In case the nodes do not provide TEE support such as when using accelerators like GPUs, secure and measured boot will 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 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
rollback: 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, as well as rollback protection. 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.

3.5. Fine Grained Policy Definitions

The previously defined policy definitions can be considered as high-level definitions. Hence, they describe the protection goals for an entire application which comprises a set of processes and services. However, it is often possible that low-level adjustments are needed such as that not all directories should be encrypted and integrity protected due to 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 aggregator process performs communication with a set of collaborator processes. In order to prevent tampering with the gradients exchanged during the learning phase, only authenticated collaborator 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., collaborator processes as well as the aggregator processes. We therefore merge high-level policy definitions with fine-grained ones in order to establish the correct properties.

We will now present the low level policy definition provided by the different security tools developed within AI-SPRINT. 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 AI application of AI-SPRINT can consist of one or more **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** is a subset of YAML, i.e., a session description is valid YAML. It is similar to, and takes its bearing from, docker-compose files. As a session description is typically stored in a file, we use session file as a somewhat interchangeable synonym for session description. We use the terms session description and session policy (or just policy) interchangeably.

```
name: $AGGREGATOR_SESSION
version: "0.3"

access_policy:
  read:
    - CREATOR
  update:
    - CREATOR

services:
  - name: aggregator
    image_name: aggregator_image
```

```
command: /usr/bin/python /root/miniconda/bin/fx aggregator start
mrenclaves: ["$MRENCLAVE"]
pwd: /workspace-demo
environment:
  PWD: /workspace-demo
  SCONE_PWD: /workspace-demo
```

images:

- name: aggregator_image
- injection_files:
 - path: /workspace-demo/cert/cert_chain.crt
 - content: \$\$SCONE::aggregator_ca_cert.chain\$\$
 - path: /workspace-demo/cert/server/agg_aggregator.crt
 - content: \$\$SCONE::aggregator.crt\$\$
 - path: /workspace-demo/cert/server/agg_aggregator.key
 - content: \$\$SCONE::aggregator.key\$\$
 - path: /workspace-demo/cert/client/col_col1.crt
 - content: \$\$SCONE::collaborator1_cert.crt\$\$
 - path: /workspace-demo/cert/client/col_col1.key
 - content: \$\$SCONE::collaborator1_cert.key\$\$
 - path: /workspace-demo/cert/client/col_col2.crt
 - content: \$\$SCONE::collaborator2_cert.crt\$\$
 - path: /workspace-demo/cert/client/col_col2.key
 - content: \$\$SCONE::collaborator2_cert.key\$\$

secrets:

- name: aggregator_key
- kind: private-key
- name: aggregator
- kind: x509
- private_key: aggregator_key
- issuer: aggregator_ca_cert
- common_name: aggregator
- name: aggregator_ca_key
- kind: private-key
- name: aggregator_ca_cert
- kind: x509-ca
- private_key: aggregator_ca_key
- export:
 - session: \$COLLABORATOR1_SESSION
 - session: \$COLLABORATOR2_SESSION
- name: collaborator1_key
- kind: private-key
- export:
 - session: \$COLLABORATOR1_SESSION
- name: collaborator1_cert
- kind: x509


```

    issuer: aggregator_ca_cert
    private_key: collaborator1_key
    common_name: col1
    export:
      - session: $COLLABORATOR1_SESSION
  - name: collaborator2_key
    kind: private-key
    export:
      - session: $COLLABORATOR2_SESSION
  - name: collaborator2_cert
    kind: x509
    issuer: aggregator_ca_cert
    private_key: collaborator2_key
    common_name: col2
    export:
      - session: $COLLABORATOR2_SESSION

security:
  attestation:
    #mode: None
    tolerate: [debug-mode, hyperthreading, outdated-tcb]
    ignore_advisories: "*"

```

Listing 3.1 - Policy Session Example for Federated Learning

The above description shows how the security tools are configured in order to generate and inject the certificates and keys necessary 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.

The above example is taken from the configuration file of the federated learning prototype. The example shows how the certificate for a certificate authority is being created (`aggregator_ca_cert`) and how it is used to sign certificates created for two collaborators (`collaborator1_cert`, `collaborator2_cert`) and one aggregator (`aggregator_ca_cert`). In addition to the certificates, also the private keys are made accessible to inside of the Docker containers through the SCONE runtime so that the Intel OpenFL implementation can use them to perform a mutual authentication itself.

3.6. Policy Server and Architecture

Figure 3.1 shows the full architecture comprising the security tools developed in AI-SPRINT. The architecture comprises three segments. The leftmost part is a SCONE-enabled data center, i.e., a data center that is equipped with the proper hardware and drivers, that can run Sconified images, and that provides a Configuration and Attestation Service (CAS). We will assume that it is managed by Kubernetes. If present, the blockchain-enabled data provider service is sconified and executed in this segment. Additionally, the sconified services communicate with one another using SCONE's Network Shield service.

An architecture view of the various components of the SCONE-enabled data center is provided in Figure 3.2. On the left side, the tools are listed needed to convert a non-confidential application into a confidential one using SCONE cross compiler or the SCONE sconify tool. In the middle, the components are displayed which are involved at runtime of an application if the provided hardware support the execution of an application in

TEEs. If no such hardware support is available, the mechanisms such as secure boot as well as measured boot will provide a minimum level of security, hence, these mechanisms are utilized.

The second segment is between the data center’s ingress node and the 5G network. This segment is responsible for isolating the traffic to and from the devices and blocking lateral movement between devices and tenants. The 5G authentication component will be implemented in the second release of the tools and will use the authentication mechanisms provided by the 5G network to configure the isolation mechanisms between the 5G network and the Sconified components, where the traffic is protected by the Network Shield. The 5G authentication component will be a Network Function running over the CHIMA framework, which is also released as part of the AI-SPRINT tools. The CHIMA application has the function of easing scalable deployment of Network Functions on high-performance programmable switches, adding performance monitoring capabilities, and automatic redeployment of Network Functions in case of performance issues. The CHIMA framework can manage multiple authentication functions and can be programmed to use other authentication mechanisms if a 5G network is not present.

A policy server takes care of pushing security configurations both to the SCONE-enabled Data Center and to the CHIMA-enabled Network.

Finally, the 5G network takes care of authenticating the devices and of moving traffic between the Edge Data Center and the devices. A more detailed architecture of the 5G network is provided in Figure 3.3.

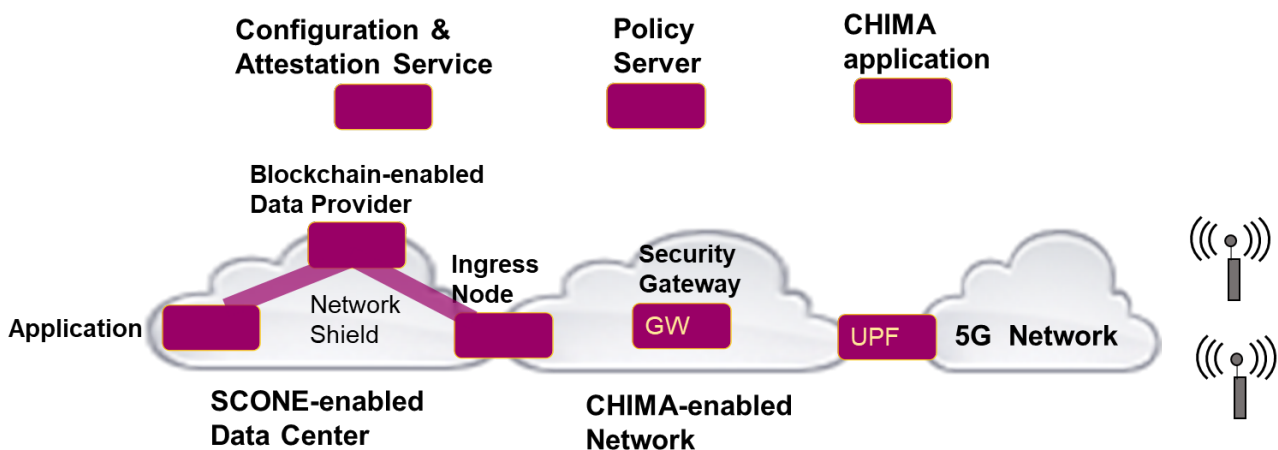


Figure 3.1 - Security tools architecture

The high-level policy definitions described in the previous section will be stored as CRR (custom resource records) using CRD (custom resource definitions) in the Kubernetes cluster the application user intends to deploy his/her application on. The use of CRD follows the cloud native approach and is the state of the art for deployments of modern applications. Note that, the policy session language for CAS is stored in a CAS instance rather than in the etcd server of the Kubernetes cluster.

Besides storing the records in the Kubernetes cluster, we will develop a **Kubernetes operator** to perform the necessary configuration steps needed to deploy an AI application in a confidential manner. The steps the Kubernetes operator will perform are the following:

- Watches for new deployments using the event bus provided in Kubernetes
- Matches labels and annotations if an application is covered by the policy definitions
- Sconifies the Docker images, i.e., turns a native Docker image into a confidential one
- Extracts and converts program arguments and environment variables

- Parses images and Python code for annotations and fine-grained security constraints
- Generates SCONE sessions and submits them to CAS
- After the deployment of pods, IP addresses will be propagated to the server instance configuring the edge network flows for the protection of collaborating edge devices

Note that the Kubernetes operator will be based largely on the Kubernetes helm post rendered we will present later in this document (see Section 4.1).

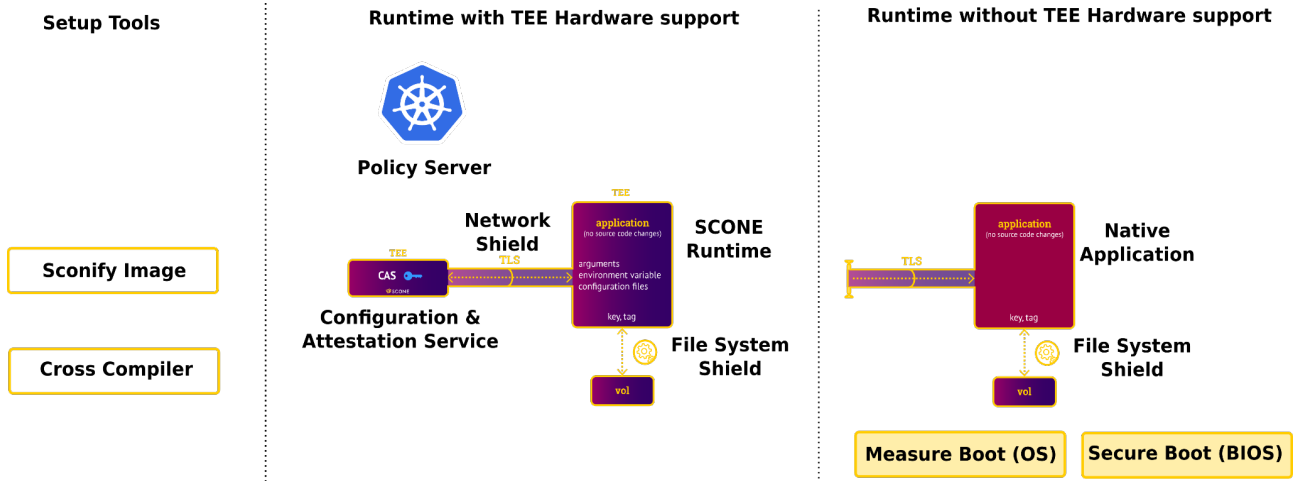


Figure 3.2 - Security tools architecture - left CLI tools, middle runtime with TEE Hardware support, right runtime if no TEE hardware support is available

The 5G technology is especially attractive for the AI-SPRINT project, because it makes it possible to provide seamless management of various deployments of the Mobile Edge Computing (MEC) resources. 5G also provides sophisticated authentication and security mechanisms that can be exploited to improve the security of applications operating across multiple layers. There is also a growing interest in private 5G deployments, particularly in the industrial context, to use the 5G technology and the 5G radio interface to communicate to Industrial IoT devices. Some countries have also licensed spectra for industrial usage to push the transition to the fully digital Industry 4.0 paradigm.

Private 5G networks can be physically isolated, or they can share resources with a public 5G network operator. In the context of AI-SPRINT, we consider a deployment scenario in which the Radio Access Network (RAN) and the Control Plane are shared between the private and the public network (see Figure 3.3). In this scenario, the telecommunications operator manages all the 5G control plane operations and operates two (or more) slices. The private slice comprises a portion of the radio spectrum, an on-premises User Plane Function (UPF), an on-premises MEC node and the corresponding computing and networking resources. The public slice comprises a portion of the spectrum, the public UPFs, the public MEC nodes with the corresponding resources, and the exit interface to the Public Data Network and the central cloud services.

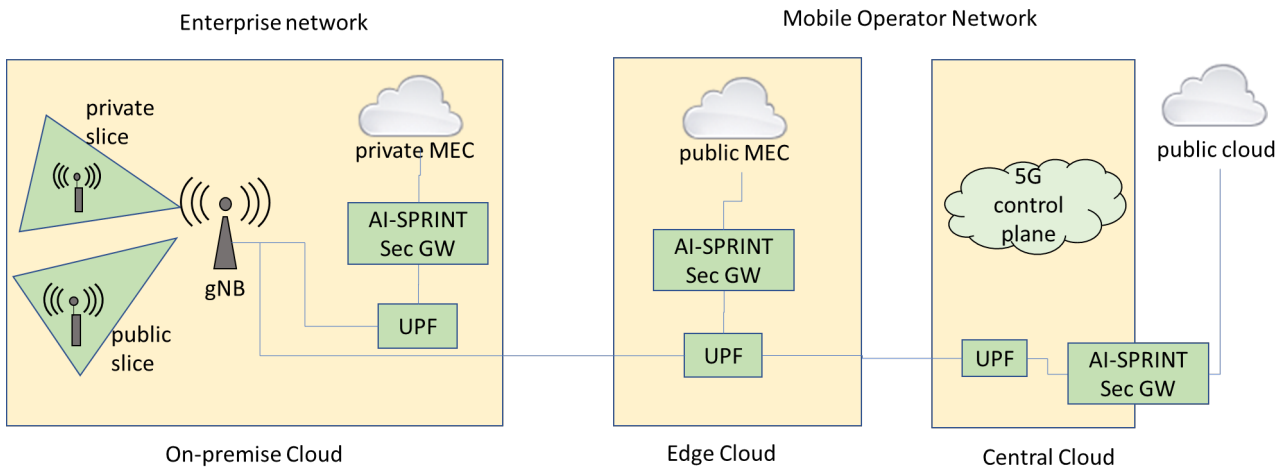


Figure 3.3 - 5G Industrial network comprising a private slice and a public slice, a private MEC, a public MEC, and a public cloud. The AI-SPRINT Security Gateway is placed between the UPF and the MEC or cloud

4. First Release of the Security Tools

This section describes the components of the runtime environment according to the Software Design Specification (SDS) standard (IEEE Standard 1016). Each component is described in terms of its external interfaces and dependencies with other components. In particular, this section focuses exclusively on the components employed as security tools within the runtime environment.

The section starts with a description of the runtime, followed by the CAS, the configuration and attestation service. Next, several CLI tools are presented necessary to turn a native application into a confidential one. Each of those components is thoroughly discussed.

Template description of components

The following template is used as the structure to provide the information for each component involved in the runtime environment. The template is included here to make this document self-contained. A similar description is also reported in AI-SPRINT deliverables *D3.1 First release and evaluation of the runtime environment*, and *D4.1 Initial release and evaluation of the security tools*.

Identification	The unique name for the component and its location in the system
Type	A module, a subprogram, a data file, a control procedure, a class, etc.
Purpose	Function and performance requirements implemented by the design component, including derived requirements. Derived requirements are not explicitly stated in the SRS, but are implied or adjunct to formally stated SDS requirements.
Function	What the component does, the transformation process, the specific inputs that are processed, the algorithms that are used, the outputs that are produced, where the data items are stored, and which data items are modified.
High level Architecture	The internal structure of the component, its constituents, and the functional requirements satisfied by each part.
Dependencies	How the component's function and performance relate to other components. How this

	component is used by other components. The other components that use this component. Interaction details such as timing, interaction conditions (such as order of execution and data sharing), and responsibility for creation, duplication, use, storage, and elimination of components.
Interfaces	Detailed descriptions of all external and internal interfaces as well as of any mechanisms for communicating through messages, parameters, or common data areas. All error messages and error codes should be identified. All screen formats, interactive messages, and other user interface components (originally defined in the SRS) should be given here.
Data	For the data internal to the component, describe the representation method, initial values, use, semantics, and format. This information will probably be recorded in the data dictionary.
Needed improvement	Description of the needed improvements of this tool with regards the AI-SPRINT 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.

4.1. Security Tools

The security tools provide several security and protection mechanisms to guarantee confidentiality, integrity as well as freshness for AI application in the context of AI-SPRINT.

Runtime Security

In this section, we will present the different security tools 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.

AI applications such as federated learning often consist of multiple **services** or **processes** that communicate via the network with each other. These services are often deployed with the help of containers. Hence, our focus is on confidential, containerized applications as well as deployments on the edge such that we can provide confidentiality for all these instances ranging from cloud to edge. The runtime security furthermore targets the mitigation of the threats listed in the previous chapter and we refer the reader to Table 3.1 listing in which way the different mechanisms cover the identified threats.

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 **VM** (Virtual Machine) 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 application¹. 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.

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 OS (operating system)
- a **container/service admin** that takes care of the services and the containers.

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

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

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

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.

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.

¹ <https://www.amd.com/system/files/TechDocs/SEV-SNP-strengthening-vm-isolation-with-integrity-protection-and-more.pdf>

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 CAS, 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 propose 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.

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:

4. 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 paused and resumed.
5. 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.

In order to avoid these cumbersome tasks, we propose 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 a Intel SGX enclave.

The following tables lists the advantages of SCONE compared to the use of the Intel SGX SDK.

Features	Intel SGX SDK	SCONE Platform
Local Attestation (LA): 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 Security
Security: Side-channel	No protection	Side-channel protection
Monitoring: SLA	-	SLA-based monitoring
Monitoring: SGX	-	SGX-resources and 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 (Visual Code)
Languages	C/C++	Most modern languages (C/C++, Python, Rust, Java, Nodejs, R, ...)
Portability	Intel SGX-specific	(eventually other CPUs)
TCO	Higher	Lower

Table 4.1 - 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 non-confidential 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.

SCONE Runtime

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	<p>The runtime manages the complete deployment of an executable/process in a so-called enclave and performs the following tasks:</p> <ul style="list-style-type: none"> ● creation/allocation of enclave memory ● loading program code and data into enclave memory ● performing attestation, i.e., creating measurement over code and data and verifying if this is correct ● performing configuration of the application ● provisioning secrets ● providing network as well as file system encryption/shielding
High level	The following image describes the high-level architecture of the SCONE Runtime,

<p>Architecture</p>	<p>including external dependencies.</p> <p>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 (the Configuration and Attestation Service) 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.</p>
<p>Dependencies</p>	<p>The SCONE runtime requires access to CAS (the Configuration and Attestation Service) in order to perform attestation as well as secret provisioning.</p>
<p>Interfaces</p>	<p>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 the application/process is being launched.</p>
<p>Data</p>	<p>The SCONE runtime uses SCONE policies as well as Environment variables to configure the application correctly.</p>
<p>Needed improvement</p>	<p>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.</p>
<p>Implemented Improvements for the First Release</p>	<p>The first release of the SCONE runtime regarding AI-SPRINT includes the following improvements:</p> <ul style="list-style-type: none"> ● Created first draft/implementation of the network shielding layer ● Improved performance with regards to system call behavior ● Integration with latest version of CAS
<p>Release Version & Repository</p>	<p>The first release version of the SCONE runtime for AI-SPRINT can be downloaded from the Docker registry: https://registry.scontain.com:5050/</p>

The objective of SCONE is to build and run applications in a confidential environment with the help of Intel SGX (Software Guard eXtensions). Although the first AI-SPRINT prototype is limited to Intel SGX, we plan to extend the support to CPU vendors other than Intel within the AI-SPRINT project.

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 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 4.1.

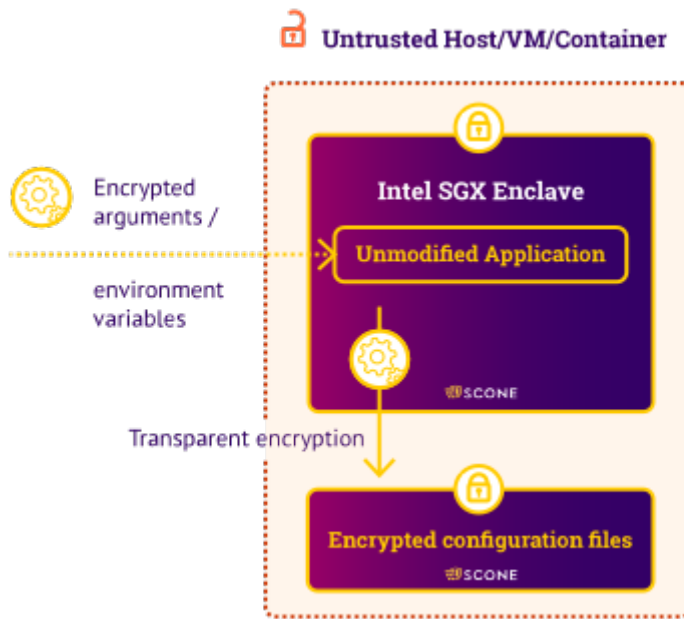


Figure 4.1 - Program arguments as well as environment variables are provided through CAS

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 4.2. 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.

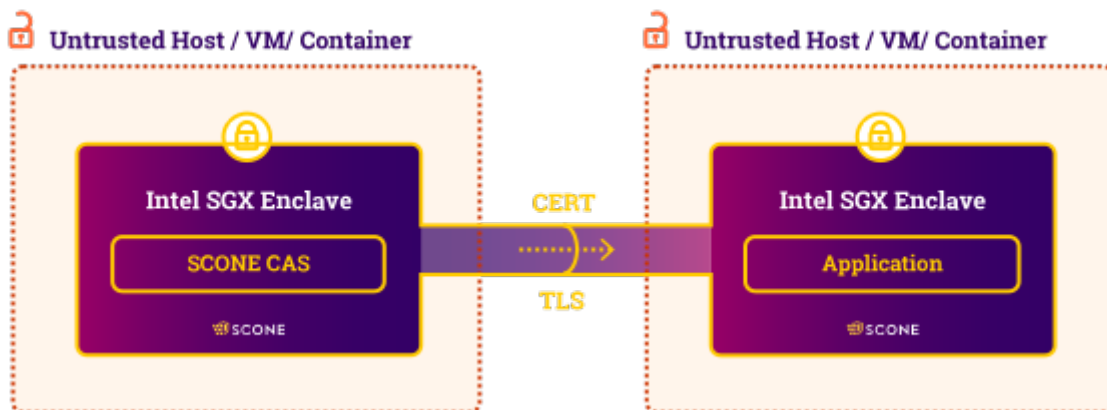


Figure 4.2 - Example where one entity is being authenticated using TLS

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 4.3. We leverage this feature in federated learning to attest if collaborators are legitimate to communicate with aggregators and vice versa.

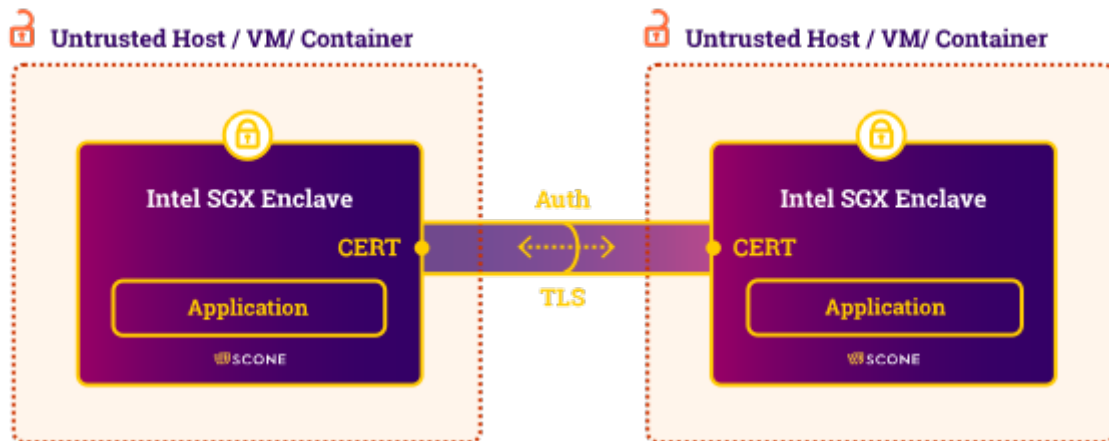


Figure 4.3 - Example about how two entities authenticate each other using TLS - mutual authentication

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.

In order to provide full protection, it is necessary that encryption keys are 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 4.4. Note that one should not use an external process for TLS termination such as stunnel.

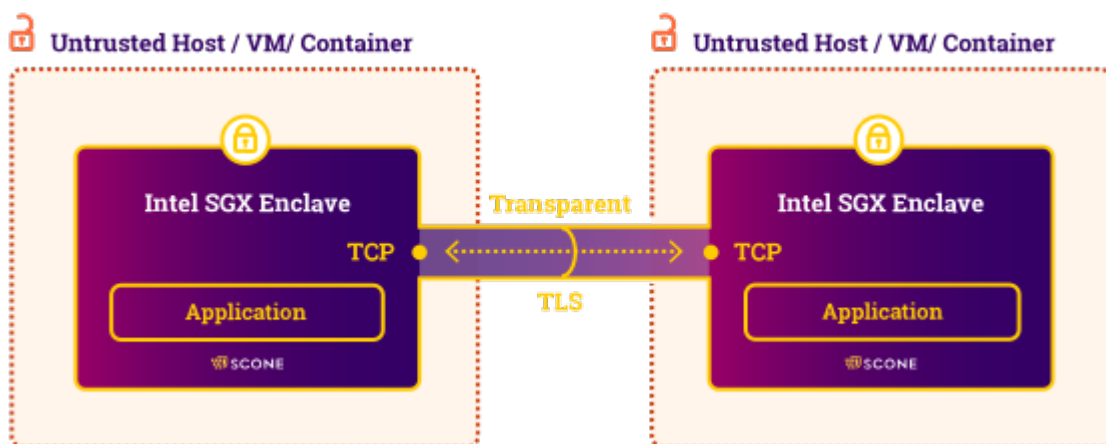


Figure 4.4 - Example of transparent network encryption. Connections are terminated inside the enclaves

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 4.5.

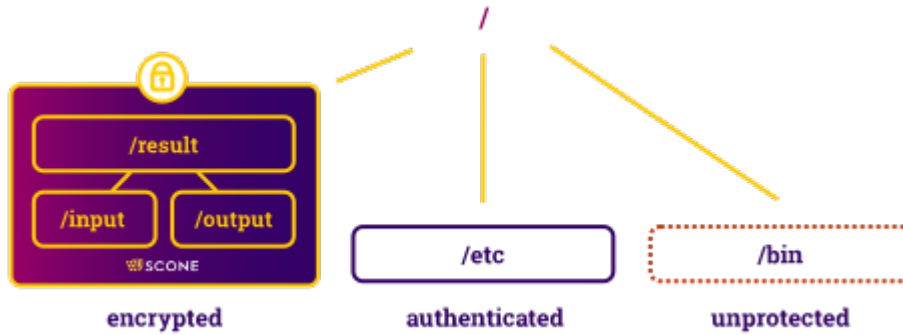


Figure 4.5 - SCONE file system shield - left volume that is encrypted, an integrity protected but not encrypted volume (middle), right no protection at all

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 application code and data into the enclave memory as well as launching the enclave.
Function	<p>The cross compiler performs compilation of source code - several languages are supported as to date:</p> <ul style="list-style-type: none"> ● C ● C++ ● Fortran ● Go ● Rust ● Python* <p>*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.</p>
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 data 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.
Implemented Improvements for the First Release	The first release of the SCONE cross compiler regarding AI-SPRINT includes the following improvements: Go support for codebases which cannot be compiled with gccgo natively. Note that Go support is needed in AI-SPRINT to harden the monitoring system described in <i>Deliverable D3.2 - First release and evaluation of the monitoring system</i> .
Release Version & Repository	The first release version of the SCONE cross compiler for AI-SPRINT can be downloaded from the Docker registry: https://registry.scontain.com:5050/

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*.

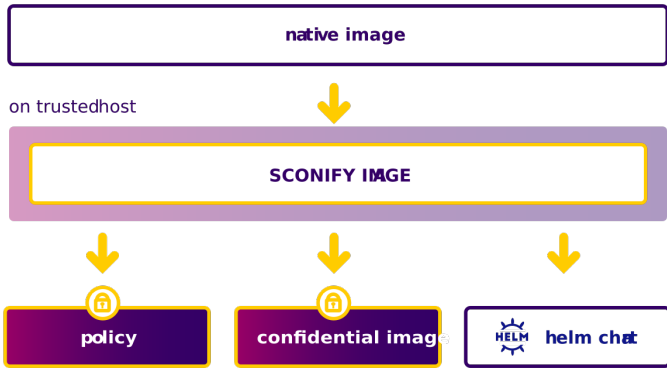
For compiled languages, our recommend approach to run an application with SCONE is as follows:

- **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**: In order 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**.

SCONE Sconify Tool

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	<p>The sconify tool re-links the SCONE libc. However, several constraints must be first met which the tool tries to do automatically:</p> <ul style="list-style-type: none"> ● 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) <p>Additional steps the tool performs:</p> <ul style="list-style-type: none"> ● 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 /usr/lib, /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 <p>Last step - Manifest generation:</p> <ul style="list-style-type: none"> ● 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.

	 <p>The diagram illustrates the workflow of the SCONIFY tool. It starts with a 'native image' box at the top. A yellow arrow points down to a box labeled 'on trustedhost' containing a 'SCONIFY IMAGE' box. From the 'SCONIFY IMAGE' box, three yellow arrows point down to three separate boxes: 'policy' (with a lock icon), 'confidential image' (with a lock icon), and 'helm chat' (with the HELM logo).</p>
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.
Implemented Improvements for the First Release	The tool did not exist before and was developed within the AI-SPRINT project.
Release Version & Repository	The first release version of the sconify tool for AI-SPRINT can be downloaded from the Docker registry: https://registry.scontain.com:5050/ ²

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	<p>The CAS performs the following tasks:</p> <ul style="list-style-type: none"> ● code attestation - local or remote ● configuration provisioning ● management of SCONE policies ● perform configuration of the target application

² This requires registration at <https://gitlab.scontain.com> in order to obtain a token for accessing the non-public docker registry.

	<ul style="list-style-type: none"> • provision secrets • configure network as well as file system encryption/shielding
High level Architecture	<p>The following image describes the high-level architecture of the SCONE Runtime as well as the interaction of CAS. Note that the SCONE runtime is weaved into the binary code of the microserver/application.</p> <p>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 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.</p>
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.
Implemented Improvements for the First Release	The first release of the SCONE CAS regarding AI-SPRINT includes the integration with the latest version of the SCONE runtime.
Release Version & Repository	The first release version of the SCONE runtime for AI-SPRINT can be downloaded from the Docker registry: https://registry.scontain.com:5050/

In the following, we will provide more details about SCONE CAS' 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.

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 also 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.

4.2. Network Security

In order to guarantee confidentiality of the data processed in AI-SPRINT, it is of paramount importance to encrypt all communications channels. Traditionally end-to-end encryption is used where both parties exchange keys to verify their identities as well as use a previously agreed encryption scheme for the subsequent communication. One way to achieve such end-to-end encryption is the widely known TLS protocol which allows to perform mutual authentication in addition to secure communication.

Unfortunately TLS support requires that the application developer uses the proper libraries as well as performs the proper configuration in order to establish the proper communication with encrypted channels. However, a huge number of applications do not support encrypted endpoints out of the box. For instance, the popular caching server **memcached** does not provide TLS endpoints, hence, all communication is unencrypted.

Several solutions exist to mitigate this problem. First, one can improve the implementation by adding the appropriate libraries such as OpenSSL etc. in order to harden the communication or use third party tools. For the third party tools, utilities such as **stunnel** can be used which essentially is a process that provides TLS termination, hence acts as a TLS endpoint that forwards the decrypted traffic to the previously plaintext socket.

Considering the cloud context, an alternative to the tools described previously such as stunnel is service meshes. **Service meshes** such as istio³ are overlay networks used in orchestration layers such as Kubernetes which also provide encryption in a similar fashion as when using VPNs. Although it does not require any code modifications from the developers side, there are several disadvantages: First, the application does not benefit from a full end-to-end encryption. It solely provides protection from malicious users outside of the overlay network. Furthermore, so-called inside attackers, i.e., malicious services inside of the network can still read and capture the plaintext network.

A complementary solution to service meshes is network encryption provided by wireless network infrastructure such as 5G. The advantage of this approach is that edge devices such as wearables, etc. used in the context of AI-SPRINT can benefit from encrypted communications although they are operating outside of a cloud environment, hence, where they cannot benefit from mechanisms such as service meshes.

An alternative to the previously described mechanism is the use of transparent network encryption as offered by the network shield by our SCONE framework.

In the following, we will detail the two approaches we have developed in the context of AI-SPRINT to harden network connections. First we will present our network shield provided by SCONE followed by the secure mechanisms provided through wireless infrastructure such as 5G we incorporated in AI-SPRINT (Blockchain-based Authentication and Authorization Mechanism and the CHIMA framework).

Identification	SCONE Network shield
Type	Functionality provided by the SCONE Runtime
Purpose	The SCONE network shield intercepts all system calls used to read or write data on a TCP socket such that all transmitted data will be transparently encrypted or decrypted.
Function	The network shield performs the following tasks: <ul style="list-style-type: none"> ● intercepting connection attempts to perform mutual authentication

³ <https://istio.io>

	<ul style="list-style-type: none"> ● encryption of written/sent data ● decryption of read/received data
High level Architecture	The network shield is part of the SCONE runtime.
Dependencies	The network shield requires access to CAS (the Configuration and Attestation Service) in order to receive certificates to perform the mutual authentication.
Interfaces	-
Data	The network shield will use SCONE policies in the near future for its configuration. Currently it uses environment variables to configure the network shield correctly.
Needed improvement	Support for CPU vendors other than Intel such that the component can run also on edge devices that for instance are based on an arm64 architecture. This will be achieved through compilation of the service using the SCONE Cross compiler which will be also implicitly used by the SCONE sconify CLI Tool.
Implemented Improvements for the First Release	<p>This is the first release of the SCONE network shield which includes the following basic features:</p> <ul style="list-style-type: none"> ● Mutual authentication through TLS handshake ● Encryption of data ● Decryption of data
Release Version & Repository	The first release version of the SCONE networkshield for AI-SPRINT can be downloaded from the Docker registry as part of the SCONE runtime: https://registry.scontain.com:5050/

SCONE Network Shield

Trusted execution environment (TEE) technologies such as Intel SGX achieve confidentiality and integrity of data processed by applications running in otherwise untrusted environments, for example public clouds. Frameworks like SCONE facilitate running container-based services in these TEEs. Most services, however, do not provide the means to enable secure network communication or authenticate clients or remote services with respect to software identity.

In this section, we will briefly present the network shield, a software layer integrated into the SCONE runtime, 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 mbedtls⁴ 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 4.6.

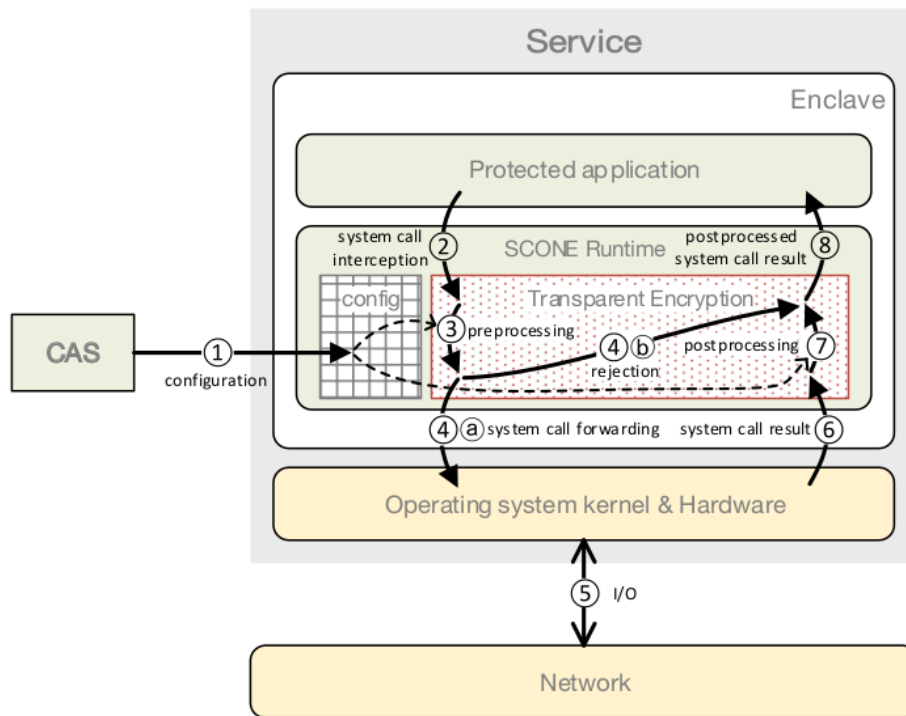


Figure 4.6 - Control flow of transparent encryption system operations

(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 <https://github.com/ARMmbed/mbedtls>

- (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).

Networks Security Mechanism at the Edge

The AI-SPRINT Blockchain-based Authentication and Authorization Mechanism at the Edge

Identification	AI-SPRINT Blockchain-based Authentication
Type	A set of microservices and a smart contracts
Purpose	The Data Provider collects data from IoT devices. When a user wants access to the data, it triggers a smart contract indicating the scope of the access. If the data owner authorizes access, the access policy is stored on the blockchain.
Function	<p>The Data Provider performs the following tasks:</p> <ul style="list-style-type: none"> ● collect data from IoT devices and write them to a database and send them to authorized subscribers ● answer queries for historical data, verifying the authorization ● enable the streaming or real time data, verifying the authorization <p>The Smart contract performs the following tasks:</p> <ul style="list-style-type: none"> ● collect requests for authorization (possibly with in-chain payments) ● store authorized policies
High level Architecture	<p>The architectural domains of the blockchain-based authentication, authorization and access control mechanism. The core component is the Data Provider, which collects data from the IoT Domain, receives the queries from the Users and verifies them against the policies stored in the blockchain.</p>
Dependencies	<p>The component requires read access to the Ethereum blockchain (or a blockchain compatible with Ethereum).</p> <p>The user and the data owner need write access to the blockchain.</p>
Interfaces	<ul style="list-style-type: none"> ● Interface to the IoT domain (any protocol compatible with RabbitMQ) ● Interface to the blockchain

	<ul style="list-style-type: none"> • Interface to the user (HTTP for queries, UDP for streaming)
Data	<ul style="list-style-type: none"> • Historical data collected from the field • Policies stored in smart contracts on the blockchain
Needed improvement	This is a novel component which has been fully developed in the context of AI-SPRINT.
Implemented Improvements for the First Release	The first release is a working prototype.
Release Version & Repository	https://gitlab.polimi.it/ai-sprint/BlockchainAuthIoT

Edge computing is particularly relevant for the businesses that operate IoT devices and Industrial IoT devices, for example manufacturing industries employing sensors to monitor the energy efficiency or the working conditions of pieces of machinery. However, the data collected over time is not only relevant for the business operating the device, but for third parties as well, for example the vendor of the equipment could be interested in gathering knowledge about how machines perform over long periods of time and how to prolong their lifespan. Other third parties could be interested in collecting and aggregating filed data to perform market predictions. When the stream of data is subject to constraints that can be checked only at the edge or the data are accessed and exchanged between devices that operate in the same geographical area, it makes sense that Authentication, Authorization and Access control are performed at the edge.

The AI-SPRINT mechanism uses blockchain technologies to provide authentication and authorization capabilities in a decentralized and transparent way. A smart contract is deployed on a public blockchain, and a signer of the contract is appointed. The business that provides the IoT data then adds policies to the contract, which are then initialized and finally signed by the customer. The AI-SPRINT mechanism also allows the late integration of additional policies by expanding the contract - to allow for a more dynamic access control management - and supports

As shown in Figure 4.7 the proposed mechanism spans multiple domains: the IoT domain, where IoT devices reside, the Edge domain, which is the core of the architecture, the Cloud domain and the Distributed domain, i.e., where the smart contract can be found on a distributed ledger. In addition, everything outside of these domains will be part of the User domain.

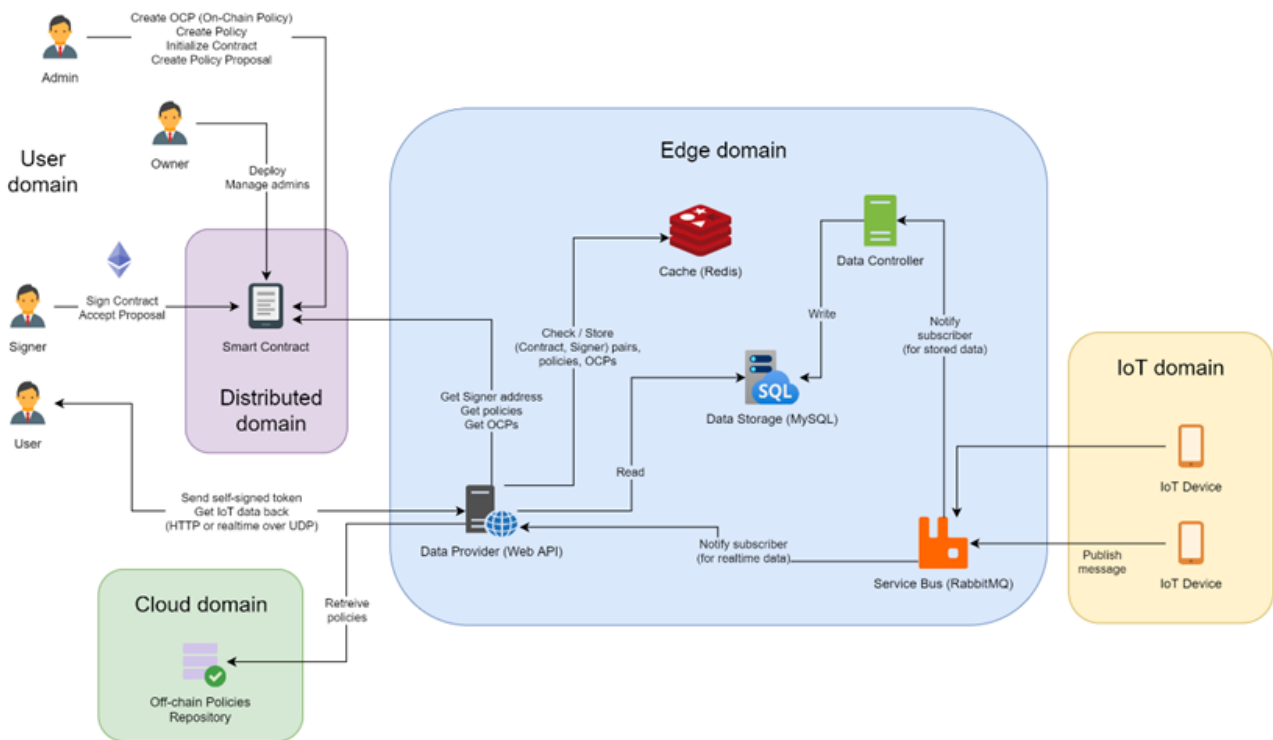


Figure 4.7 - Architectural domains of the blockchain-based authentication, authorization and access control mechanism

The **IoT domain** groups all the IoT devices that periodically generate data. These devices interact exclusively with the Service Bus component in the Edge domain in order to write data to the message queues. The written data can be of any kind. For the sake of simplicity we only present the mechanism with reference to json-serialized strings. The Service Bus should support several protocols, in order to be compatible with as many IoT devices as possible.

The components of the **Edge domain** need to provide an ingress endpoint where the IoT data comes in, and an egress endpoint where the IoT data can be fetched, after being processed and after the authentication and access control policies have been applied. The Service Bus is the ingress endpoint; it implements the pub/sub design pattern, which splits the connected clients into two categories: the ones that provide data (publishers) and the ones that consume the data (subscribers). In this case, the publishers are the IoT devices, while the subscribers are the Data Controller and the Data Provider. The Data Controller consumes messages from the queues and creates queries to store them in the Data Storage. It needs read access to the Service Bus and write access to the Data Storage. This component should be up at all times, and there can be multiple replicas of it, since messages can only be consumed by a single Data Controller and are then deleted. The Data Storage is where we IoT data is stored over periods of time to allow asynchronous queries. The Data Provider is the egressing endpoint where IoT data is delivered to customers, according to the specifications of the Smart Contracts that they signed. This component comprehends two servers: an asynchronous one for answering queries, and a streaming one, which delivers data in real-time. In both cases, authentication and access control policies are applied before serving data from the endpoints.

As previously discussed, the **Cloud domain** is used to store resources that do not need to be accessed very often, due to the high access cost in terms of network delay in real-time applications. We can use this domain to store off-chain policies: in fact, since this architecture provides the ability to cache policies, they only need to be read and verified once. After being verified, since by design they cannot be modified, we assume the cached version is always valid.

The **Distributed domain** is where the blockchain is situated. Nodes that make up the block-chain network can be anywhere, as long as they are synchronized with each other. The users and the service provider can each run their own node, and refer to that node to process transactions and fetch data from the blockchain.

The users in the **User domain** belong to four categories. Owners are special administrators that are able to deploy valid contracts to the blockchain. Only contracts that are deployed by addresses that belong to the whitelist of owners are to be considered valid, otherwise anyone could forge a contract and deploy it to the blockchain, claiming to have purchased the service. Additional administrators can be appointed by any existing administrator to manage contracts on their behalf. The third category is made of users who sign the smart contracts and pay for the service: we will call them signers. During the signing process, a signer can provide the public key of the actual user that will fetch the IoT data from the Data Provider, according to what was allowed in the contract that they signed.

Authentication Example

Authentication is implemented in the Data Provider module and it is in charge of checking a few requirements. First of all, it verifies that the user identity matches their claims. This is done through an authentication token that is sent by the user. The token itself is self-signed because the information about the identity of the user is stored inside the smart contract and can easily be verified. Alongside that, the authentication module checks that the address of the contract is valid, meaning that it is signed and it references the user identity. The final step is to check that the contract was deployed by an authorized account in the blockchain. This prevents contract forgery attacks, in which a user deploys and signs its own contract on the blockchain to get access to IoT data without paying the service provider. The full exchange is depicted in Figure 4.8.

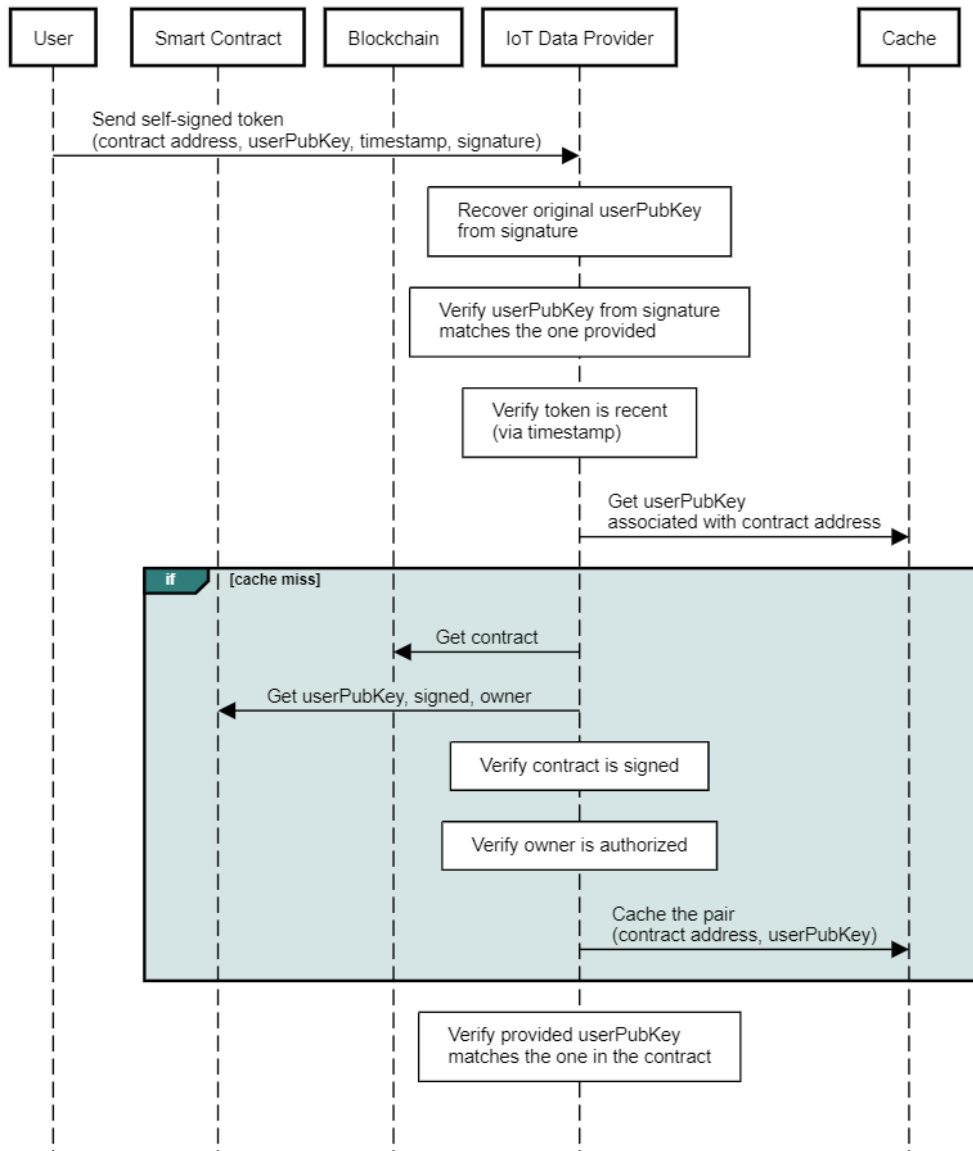


Figure 4.8 - User authentication

Authorization Example

Authorization is also implemented in the Data Provider. This module validates each query against the policy defined in the smart contract. Policies are identified by a *resource* field, which uniquely identifies the endpoint that the user is querying. For example, a query to the following URL

`http://provider.com/temperature/latest`

represents an attempt to retrieve the resource named *temperature/latest* and a policy bound to the same name must be present in the smart contract in order to allow access. In addition to the resource name, users can also supply parameters in the query, such as the ones in the following example

`http://provider.com/temperature/latest?count=10&device=Sensor_1`

These parameters will be checked against the ones inside the body of the policy. This mechanism is widely known as ABAC (Attribute-Based Access Control) and it provides the highest granularity, since it allows providers to have full control on the data that each user can access. The full exchange is given in Figure 4.9.

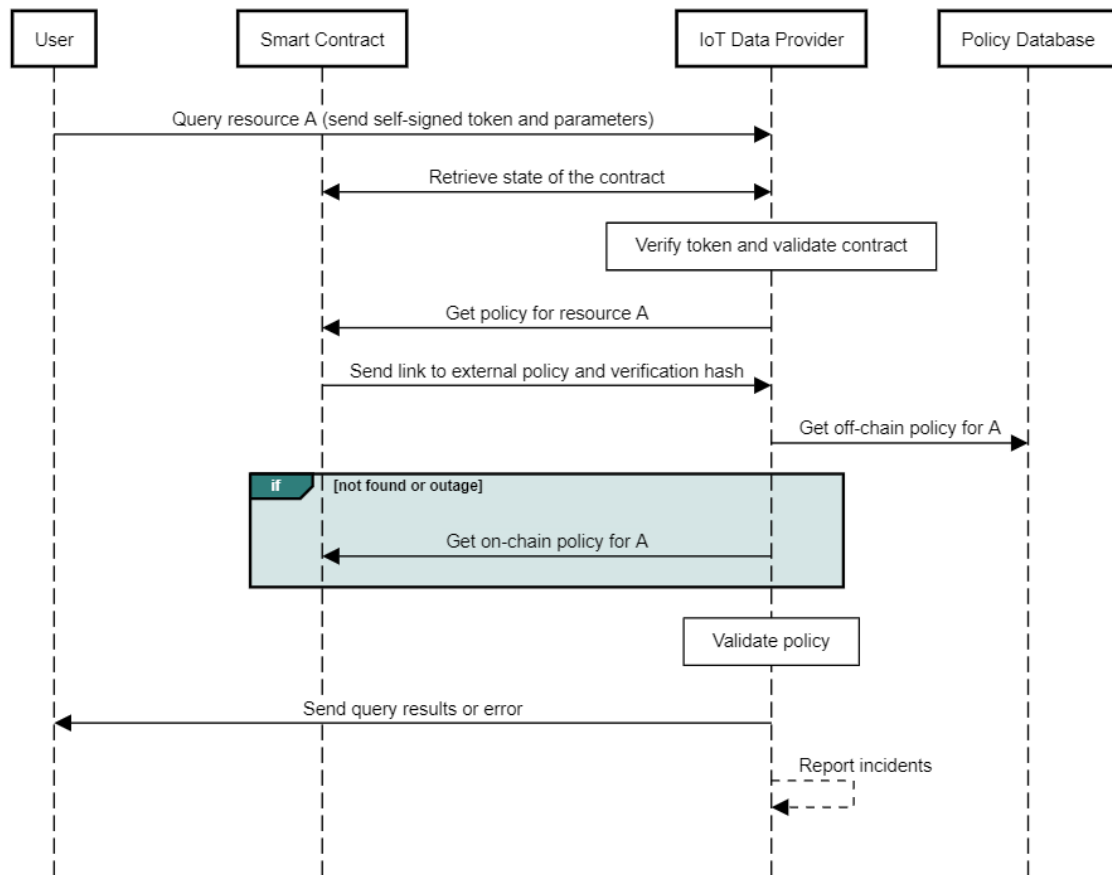


Figure 4.9 - Authorization of a user query

The CHIMA (CHain Installation, Monitoring and Adjustment) framework

Identification	CHIMA (CHain Installation, Monitoring and Adjustment)
Type	A controller/orchestrator for deploying network functions on heterogeneous hardware. An extension to network protocols for passive real-time monitoring of network delay.
Purpose	The component takes as input a list of service requests, which include a sequence of network functions and of performance requirements. The component chooses how to deploy the network functions on the available hardware and takes care for routing the packets through the correct functions. The monitoring protocol collects delay measurements for each packet. In case of an excessive delay, the component moves the functions and reroutes the traffic.
Function	<ul style="list-style-type: none"> • Choose the path that meets the performance requirements • Deploy multiple functions in the same node and deploy the correct routing rules • Collect the delay measurements • Redeploy the functions in case of performance issues

<p>High level Architecture</p>	<p> : REST API : P4Runtime gRPC → : Communication </p> <p>The CHIMA application orchestrates the deployment of Virtual Network Functions both on general purpose hardware, through the CHIMAClient, and on programmable hardware using the P4 language. Deployment on P4 hardware is performed through ONOS. ONOS also deploys routing information.</p>
<p>Dependencies</p>	<p>The component requires the ONOS controller and hardware and software switches supporting the P4 programming language</p>
<p>Interfaces</p>	<ul style="list-style-type: none"> ● Management interface
<p>Data</p>	<ul style="list-style-type: none"> ● Performance measurements
<p>Needed improvement</p>	<ul style="list-style-type: none"> ● Enhancement of redeployment in case of performance issues ● Extension to support performance monitoring across functions implemented in languages different from P4 ● Integration of telemetry with the AI-SPRINT monitoring subsystem ● Development of the 5G authentication function
<p>Implemented Improvements for the First Release</p>	<ul style="list-style-type: none"> ● Redeployment in case of performance issues ● Support for performance monitoring across functions implemented in languages different from P4
<p>Release Version & Repository</p>	<p>https://gitlab.polimi.it/ai-sprint/CHIMA</p>

The use of Virtual Network Functions (VNFs) for the deployment of security functions such as firewalls and deep packet inspection, makes it easier and faster to manage their provisioning. However, this introduces a tradeoff between flexibility and performance. Executing packet processing logic on regular CPUs is less efficient, both in terms of throughput and power consumption.

Recent advancements in the field of Programmable Data Planes, and In-Network Computing in particular, showed that offloading sections of these services to programmable switches is a viable way to eliminate the tradeoff, bringing back the processing performance to the level that was offered by specialized hardware middleboxes.

In addition, the ability to define arbitrary logic with the P4 language enables the development of other features alongside regular forwarding and processing, such as the real time monitoring of flows with In-band Network Telemetry (INT).

Such techniques can be exploited for more than diagnosis or logging, since real time feedback on the performance of a service could allow an orchestrator to take immediate action in response to congestion or faults. The CHIMA (CHain Installation, Monitoring and Adjustment) framework introduces the possibility of specifying the performance requirements, such as the maximum latency and the maximum jitter, either for sections of the service or for its entirety. A system based on real time monitoring of the service's communications with In-band Network Telemetry is proposed to maintain the requested level of performance throughout its entire lifetime. This includes the detection of adverse events and their solution through the rerouting or redeploying of affected components. A prototype of CHIMA has been developed and used to evaluate the feasibility of the proposed solution.

The proposed framework, shown in the Figure 4.10, is designed to operate on a network built with programmable switches that can be targeted by a P4 compiler, while the computing nodes are capable of running containers. The ONOS SDN controller is used to manage the forwarding of the packets; the computing nodes are configured by using Docker.

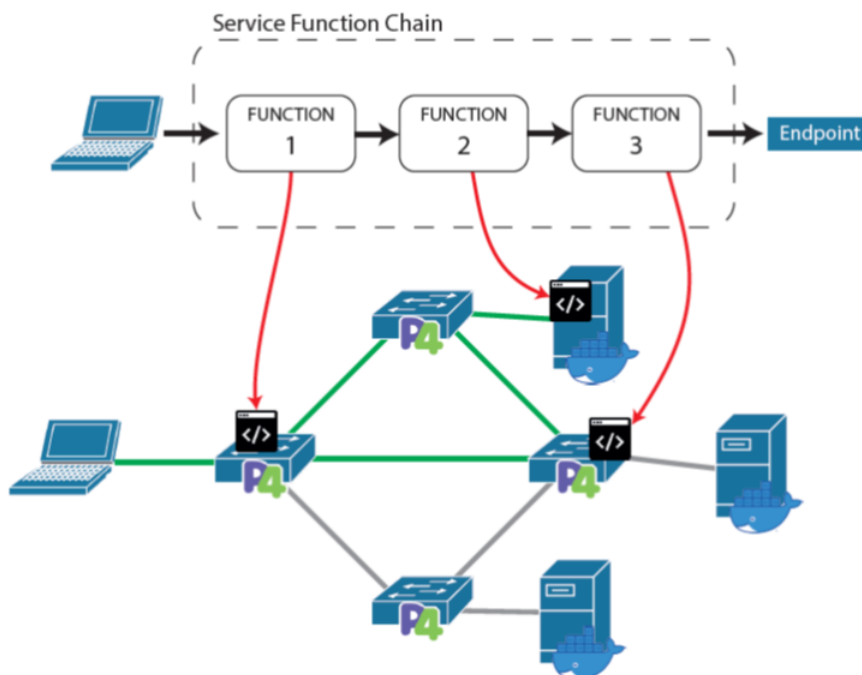


Figure 4.10 - An heterogeneous service chain addressing both P4 programmable switches and Docker containers

The CHIMA framework consists of multiple components, distributed over a supported network as shown in Figure 4.11.

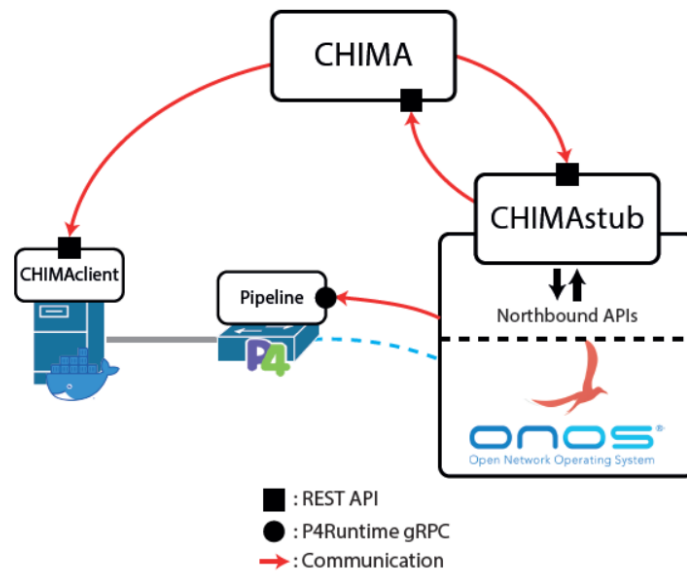


Figure 4.11 - The components of the CHIMA framework

CHIMAstub is an application running on top of the ONOS controller. It exposes topology information and events and allows the interaction with the network devices through an extension of the ONOS REST APIs.

CHIMAcient is an agent that runs on hosts and applies the header stacks for the correct routing of packets of managed services.

The P4 pipeline is installed on all the switches. In addition to providing basic forwarding, it supports In-band Network Telemetry according to the INT v1.0 specification. In a future release, the collected telemetry will be integrated with the AI-SPRINT monitoring infrastructure (Deliverable 3.2). The collected measurements are used as the base for the inclusion of user provided VNFs targeting the P4 platform.

The CHIMA module is the central manager of the system. Its tasks are to build and maintain an internal representation of the network topology, to collect INT data from the reports delivered by switches, to compute a deployment strategy based on the available topology information, and to perform the deployment of functions and manage their routing.

Routing is based on Segment Routing over MPLS (SR-MPLS). In particular, CHIMA uses the approach defined by RFC8660, in which Segment Identifiers are represented as MPLS labels. An example topology showing the packet headers is shown in Figure 4.12.

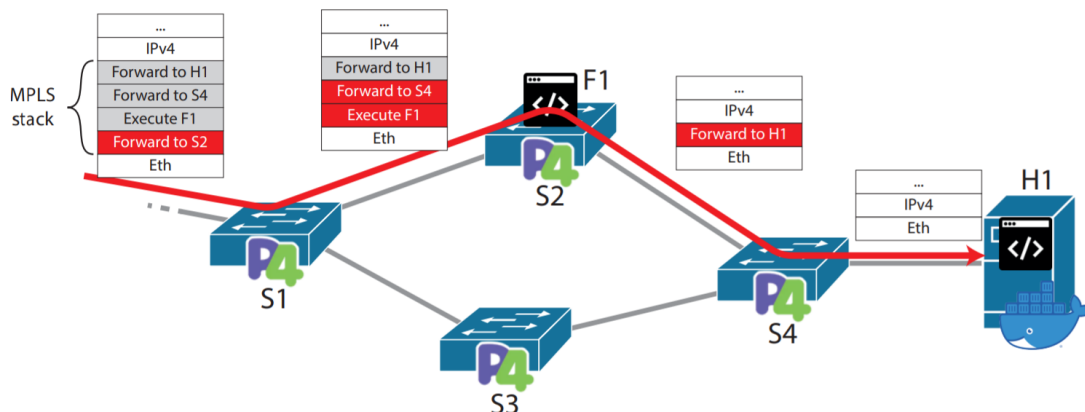


Figure 4.12 - Usage of SR-MPLS in CHIMA

5. Performance Evaluation

In this chapter, we will present various performance evaluations we executed in order to assess the overhead introduced through our approach. We will first present the runtime related performance results followed by an evaluation of our network shield we introduced in AI-SPRINT.

5.1. 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.

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 with each other 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 [30] and (ii) MNIST handwritten digit dataset [31]:

- **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 [32,33].
- **MNIST:** The MNIST handwritten digit dataset[31] consists of 60000 28 pixel images for training, and 10000 examples for testing.

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 the 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.

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 the communication among worker nodes (in a distributed training process). To achieve this 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.

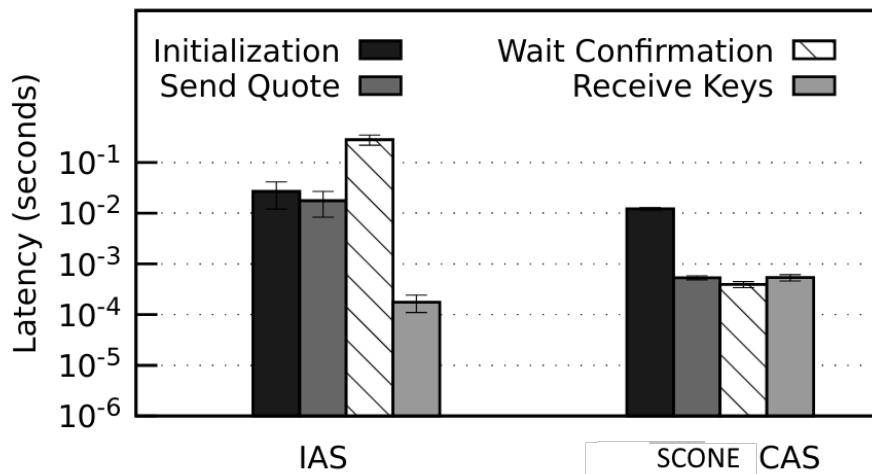


Figure 5.1 The attestation and keys transferring latency comparison between TensorFlow with the traditional way using IAS

Figure 5.1 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 ~ 280 ms. In total, our attestation using CAS (~ 17 ms) is roughly 19 \times faster than the traditional attestation using IAS (~ 325 ms). 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.

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 [34]. Graphene is an open-source SGX implementation of the original Graphene library OS. It follows a similar principle to Haven [35], 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 conduct this experiment, we use the Desktop machine to install Ubuntu 16.04 since the Graphene based system does not work with Ubuntu 18.04. 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: `$ label_image -m mod.tf lite -i inpu.bmp -t 1`. For the latency measurement, we calculate the average over 1,000 runs. We use a single bitmap image from the Cifar-10 dataset as an input of evaluated systems.

Models. For classifying images, we use several pre-trained deep learning models with different sizes including Inception v3 [37] with the size of 91MB, Inception-v4 [36] with the size of 163MB and Densenet [38] 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.

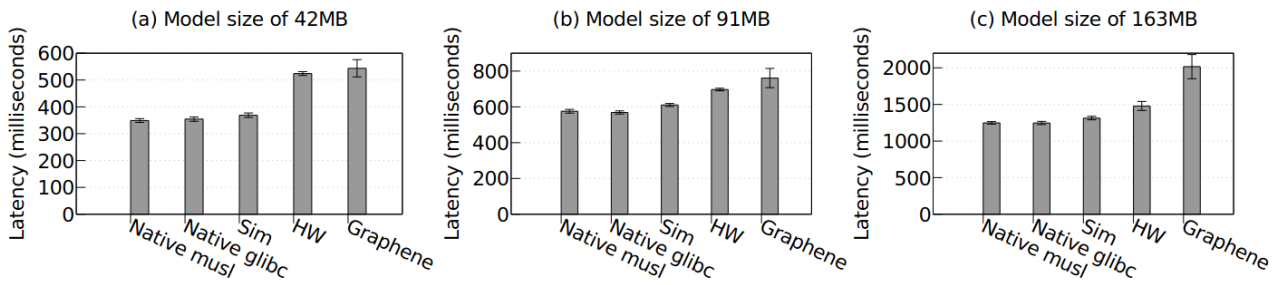


Figure 5.2 - Comparison between TensorFlow, native versions and the state-of-the-art Graphene system in terms of latency with different model sizes, (a) Densenet (42MB), (b) Inception_v3 (91MB), and (c) Inception_v4 (163MB)

#1: Effect of input model sizes. Figure 5.2 shows the latency comparison between TensorFlow with Sim and HW mode, native TensorFlow Lite with glibc, native TensorFlow Lite with musl libc, and Graphene-based system. TensorFlow with Sim mode incurs only $\sim 5\%$ 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 $1.39\times$, $1.14\times$, and $1.12\times$ 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 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 ($\sim 94\text{MB}$). In particular, with the model size of 42MB, TensorFlow with HW mode is only $1.03\times$ faster compared to Graphene-based system, however, with the model size of 163MB, TensorFlow with HW mode is $\sim 1.4\times$ 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.

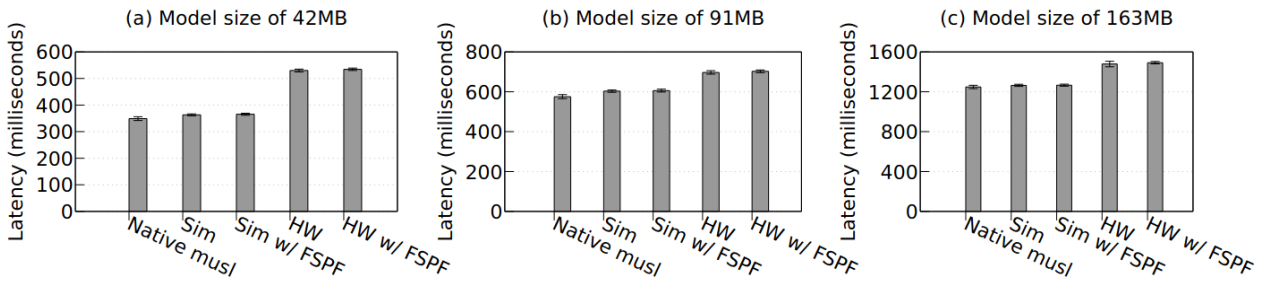


Figure 5.3 - The effect of file system shield on the classification latency with different model sizes, (a) Densenet (42MB), (b) Inception_v3 (91MB), and (c) Inception_v4 (163MB)

#2: 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 4.3 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.

Continues 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. These numbers are also reported in the AI-SPRINT *Deliverable D7.5 1st year progress report* for the KPI *K4.1 Performance overhead of Code Execution Security ≤30%*. *The current value we achieved at M12 is 47%* for our continuous runtime benchmark.

5.2. 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 Web servers. 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

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.

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.

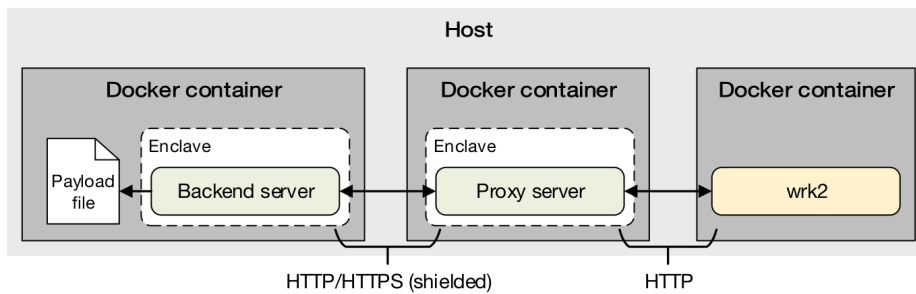


Figure 5.4 - General evaluation benchmark setup

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-

⁵ <https://github.com/giltene/wrk2>

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.

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 20% of recorded achievable throughput rates, in an automated fashion.

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.

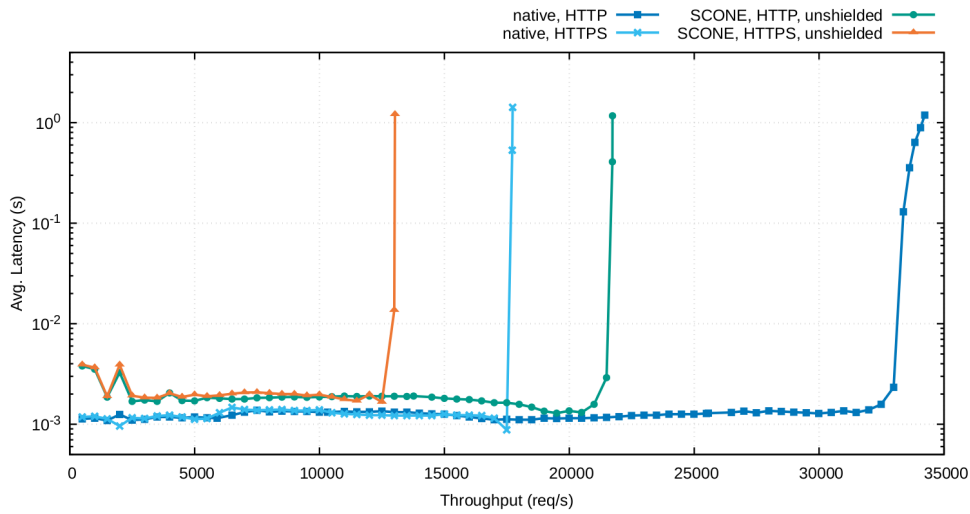


Figure 5.5 - Network Shield latency/throughput development for native and unshielded SCONE HTTP(S) configurations: nginx, 64 KiB payload, hw mode

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 5.6 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% which we also report as KPI K4.4 in the AI-SPRINT *Deliverable D7.5 1st year progress report*. 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).

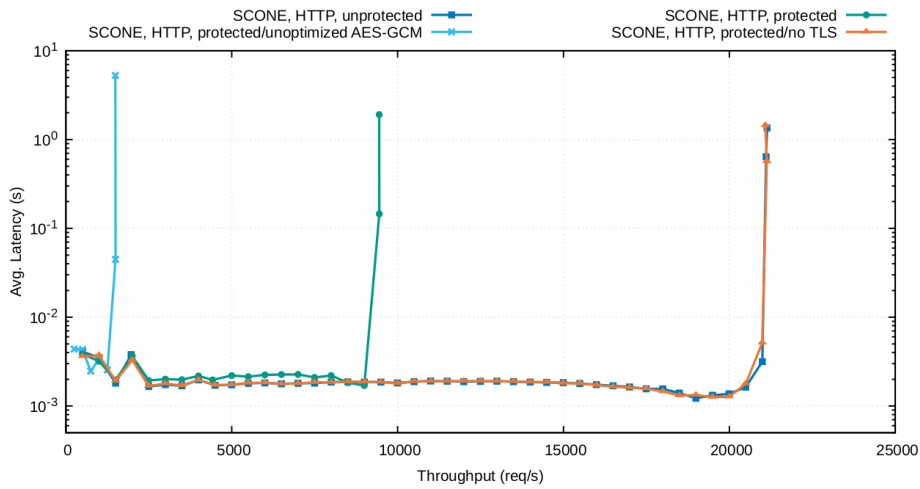


Figure 5.6 - Network Shield latency/throughput development for unprotected, protected and TLS-over-TLS unwrapped HTTPS configurations: nginx, 64 KiB payload, hw mode

5.3. Evaluation of the CHIMA Framework

We evaluated the CHIMA framework over the topologies and services shown in Figure 5.7. The test cases have been simulated with the FOP4 platform [39], using software switches. To evaluate the detection and redeployment performance, the framework has been instrumented to record the timestamps of relevant events. All measurements have been performed on a bare-metal installation of Ubuntu 20.04 LTS, running on an Intel Core i7-6700 CPU with 64GB of RAM.

Each experiment starts with the creation of the simulated topology on FOP4, followed by the deployment of the related service with CHIMA. After the service is properly set up, the latency of the targeted link is artificially increased to a level that causes requirements to be exceeded. The event is detected when one of the path-wise measurements reaches the value of the requirement, and causes the redeployment process to begin.

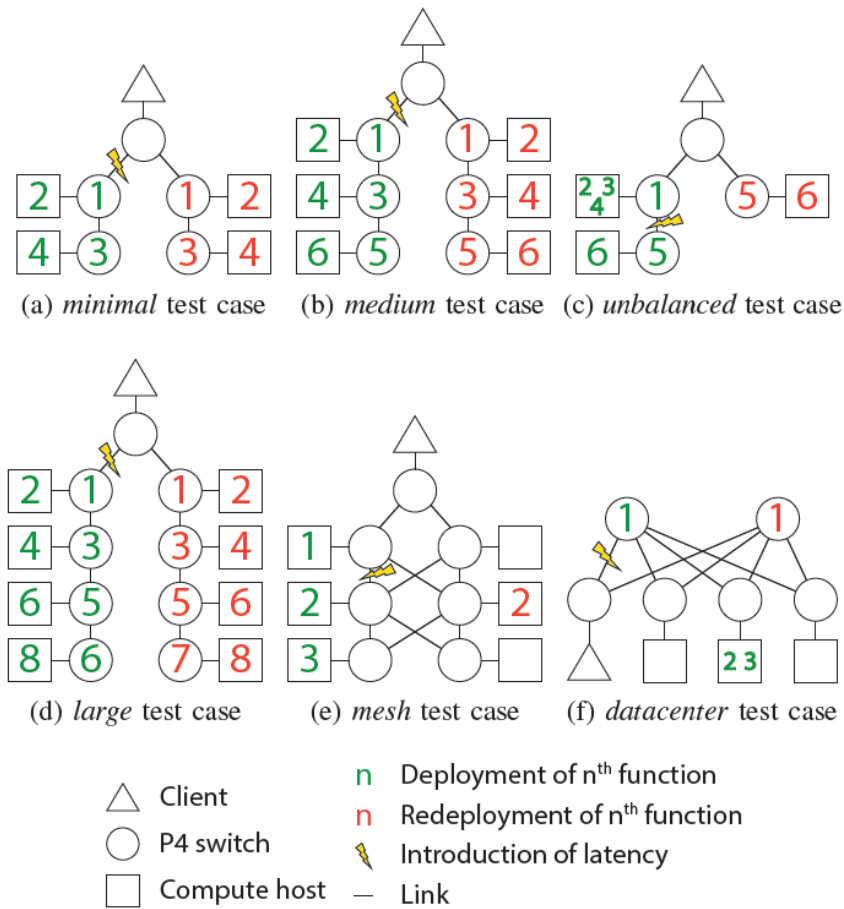


Figure 5.7 - Topologies used to evaluate the redeployment times

The first set of measurements have the objective of determining how long it takes the framework to detect the introduction of a perturbation, depending on the values of user-configurable parameters.

The detection delay is computed as the time CHIMA takes to detect an exceeded requirement after the first packet of a perturbed application is sent. Assuming the properties of topology and service to be constant, we can consider the detection delay to be a function of the polling interval and the EWMA coefficient. For this reason, all the measurements of this section have been performed on the minimal topology, shown in Figure 5.7 (a).

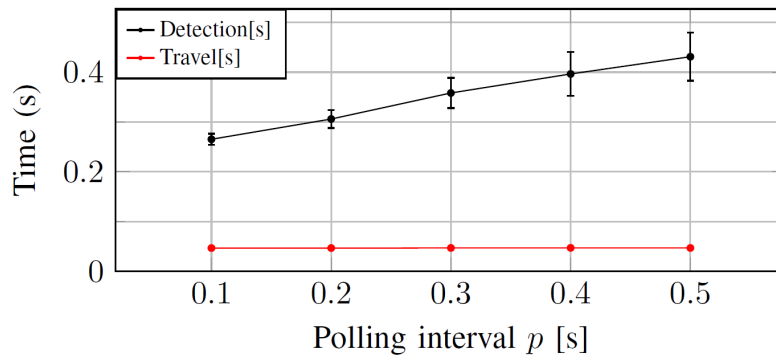


Figure 5.8 - Delay in the detection of an exceeded requirement with different intervals for the polling of new measurements from the INT collector. The bars indicate the 95% confidence interval

Figure 5.8 shows the average detection delay versus the rate at which the userspace component of the eBPF INT collector polls new EWMA values. For comparison, the Figure also shows with a red line the time taken by a request to traverse the function chain end-to-end. As expected, the results present a clear linear trend, directly proportional to the polling interval, p . A constant contribution is given by the time needed for the EWMA-smoothed measurements to cross the threshold, while the slope of the curve is due to the polling frequency. As expected, the mean delay due to the polling latency is about $p/2$.

The second parameter is the smoothing coefficient α for the computation of the Exponential Weighted Moving Average (EWMA). Larger values of α result in more weight given to recent data rather than the old average. This is clearly shown by Figure 5.9, in which smaller coefficients cause the time needed for convergence to the new measured values to grow exponentially. This data confirms the effectiveness of α to tune the response of the framework in case of short-lived congestion events. Therefore, the optimal value for this parameter should be determined based on the intended application.

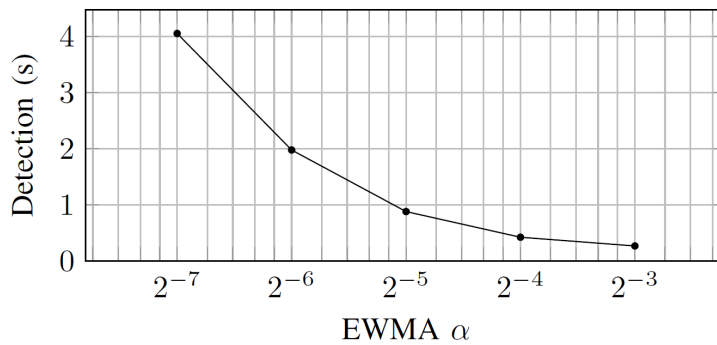


Figure 5.9 - Delay in the detection of an exceeded requirement with different coefficients for the computation of the EWMA on link measurements. The bars indicate 95% confidence intervals

Another crucial measurement to outline the framework performance is the time needed to complete a redeployment. Table 4.2 presents a comparison of the relevant characteristics among the test cases for which redeployment times have been measured.

Topology	P4 (s)	Containers (s)	Paths (ms)	Metadata (ms)
mesh	-	1.13	54.23	2.68
data center	5.20	-	288.60	2.79
unbalanced	5.19	0.97	60.57	2.81
minimal	6.06	1.50	48.79	3.50
medium	6.93	2.12	125.89	4.81
large	8.28	2.54	477.55	6.10

Table 5.1 - Breakout of redeployment times for different topologies. 95% CI

Table 5.1 shows the latency of the various tasks contributing to the redeployment delay. Since the redeployment of the different components is executed in parallel, the total time will be dominated by the slowest one. The installation of P4 functions proves to be the dominant factor, causing the total time to be in the order of seconds. The two contributions to this delay are the reconfiguration of the switch’s pipeline and the reinstallation of the correct set of rules in the pipeline’s tables. While the former is due to the use of software switches and could be reduced to tens of milliseconds with vendor specific features, the latter is caused by ONOS management of programmable data planes, which is not structured for time sensitive pipeline changes. Improvements to target this specific use case could drastically decrease delays. Times for path distribution and metadata adjustment, which can be entirely attributed to the framework logic, are much less significant than previous ones, adding minimal overhead.

With respect to the project KPIs, reported also in the AI-SPRINT Deliverable D7.5 1st year progress, the CHIMA framework adds negligible delay except in case of redeployment. Therefore, the impact on KPI K4.3 (Network additional latency) can be considered negligible. On the other hand, the framework adds some network overhead in terms of additional packet headers to carry the telemetry and the information to route the packet to the correct Network Functions. The target value for K4.4 (Traffic overhead due to Network Security policies) is 10% or less. The CHIMA framework adds, on average, 52 bytes per packet. Considering an average packet of 512 bytes, the resulting overhead is about 11%, which is the current value of the KPI at M12.

5.4. Evaluation of the Blockchain Authentication Mechanism

When dealing with blockchain applications, it is important to consider that there is an intrinsic overhead in the time domain due to the fact that transactions need to be mined inside a block before being usable. This depends on the average time (called block time) necessary to mine a block in the blockchain that is being used, and also on the amount of gas offered to the miners. In fact, the more gas is sent, the more willing a miner will be to include the transaction in the next block, speeding up its validation. If a low amount of gas is sent, there could be a long period of time before the transaction is finally included in a block. Ideally, it would be best to wait for one or more confirmations, i.e. blocks added after the one in which the transaction was written, to prevent the possibility of using blocks fabricated by malicious actors. The block time, in turn, depends on the difficulty of finding a valid hash for the block that is being mined. As of July 11th, 2021, the average block time for the Ethereum mainnet is around 15 seconds, while for the Binance Smart Chain mainnet it is around 3 seconds. In the proposed solution, transactions only need to be performed in order to manage the smart contract (for example by adding policies) and not to validate IoT data, so some delay is tolerable.

In addition to the time needed to write data to the blockchain by performing transactions, the time it takes to access data should also be taken into consideration. When there is a need to access data on the blockchain, assuming the node is already synchronized with the latest block, the time needed to perform the read operation depends on the node and its indexation of blocks. For Ethereum, web3 is the de facto standard for interacting with nodes in the blockchain. The web3 provider is in charge of communicating with a node,

through which it fetches trustworthy data from the blockchain. This provider can be deployed autonomously on a local node, or can be accessed via a third-party web service. A malicious third-party provider can manipulate the data that is read from the blockchain, but when it comes to writing data, a malicious provider can only drop a transaction, not alter it, since it is extremely difficult to produce a valid signature for the modified data without having access to the private key of the user.

Table 5.2 shows the average time it takes to fetch the bytecode of a contract from the Kovan test chain for the popular third-party *web3* provider *Infura* using a regular consumer-grade internet connection.

Request	Average Time (ms)
First Request	450
Other Identical Requests	116

Table 5.2 - Average time to fetch a smart contract from the blockchain

Infura automatically caches the responses for identical requests that belong to the same session. Even if we don't consider the first response, it still takes more than 100 ms to get the contract from the blockchain. Another important consideration to make is that this request will only retrieve the bytecode of the contract. If there is a need to read values stored inside the contract, additional requests are needed. This overhead quickly builds up and it might even take a few seconds to fetch all the data from a big contract: this shows that caching is very important for this application. Setting up a cache inside the Edge node lets us avoid the overhead of fetching the same data from the blockchain every time.

With respect to the project KPIs, reported also in the AI-SPRINT Deliverable D7.5 1st year progress, the KPI K4.3 (Network additional latency) has a target value of 10% or less. Considering a sensor with a data rate of 1 sample per second, we have an additional latency of 45% for the first request and of 12% for the following ones, which is the current value of the KPI at M12.

6. Towards the Integrated framework for the Security Tools

6.1. Integration plan for the Security Tools

The integration plan of AI-SPRINT aims at harmonizing the definition of the requirements with the design and development phase and ensuring that scientific and technical activities comply with the use cases definition. In WP1, a specific task is devoted to these integration activities and has also a design side, though it is confined at the level of architectural design. Figure 5.1 depicts the validation strategy of AI-SPRINT through milestones defined in the work plan of the project.

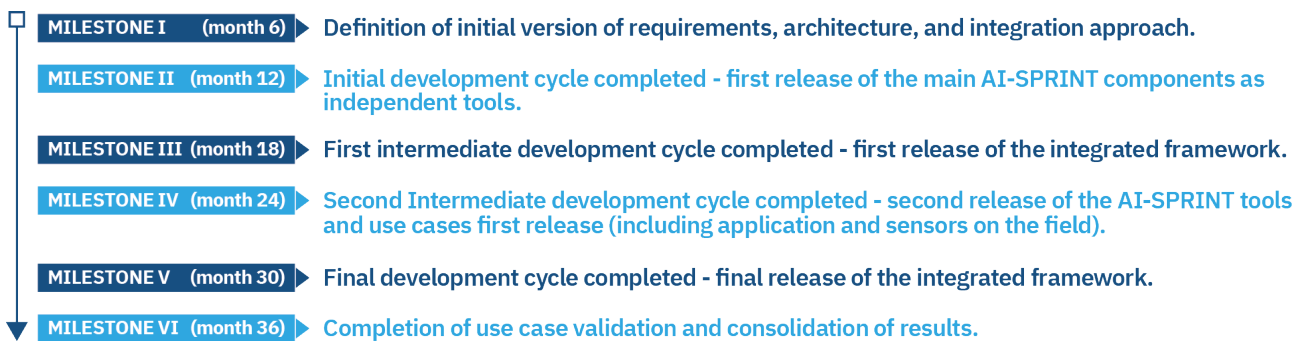


Figure 5.1 - Milestones for components development

This deliverable realizes the Milestone II which releases the first version of the individual components related to security and according to the development plan described in *D1.2 - Requirements Analysis*. In particular, this document describes the:

- Development of the first release of the security tools including the definition of security policies, their definition and architecture with regards to the server and entities managing these policies.
- Initial release of the security tools (this covers the different components we developed within AI-SPRINT such as improvements with the SCONE cross compiler, the creation of the sconify tool as well as the file system and network shielding providing transparent encryption for data at rest and in transit). An initial evaluation of the network shielding is provided as well.

At M18 we will release the integrated framework which will include a first integration of the WP4 components with the different applications used with AI-SPRINT, in particular it will report on the initial integration of SCONE within the federated learning application to over **Task 4.1 Data Security** as well as other components such as the monitoring component carried out through InfluxDB etc. Within the integration, the different AI-SPRINT runtime components will be sconified, i.e., such that they can harness TEEs and run in enclaves as well as pre-configured such that attestation and secret injection are performed while these applications are running. These actions will be executed in the light of **Task 4.2 Application Execution Security**.

With regards to **Task 4.3 Network Security**, we will provide configurations in a semi automatic fashion such that TCP channels will be transparently encrypted for services deployed on the edge as well as on cloud. Furthermore, this approach will be complemented with the use of service meshes. We will also incorporate

the policy management into the appropriate components starting at the top, i.e., the Kubernetes cluster up to the SCONE runtime.

To ensure continuous delivery as well as harden security for devices that do not offer enclaves, we will implement a first version of secure code compilation as part of **Task 4.4 Patch Management and Secure Boot**. This will also include a first prototype for operating system attestation as part of the secure and measured boot mechanisms we plan to integrate.

7. Summary & Conclusions

This deliverable has provided the initial version of the security tools developed for the AI-SPRINT platform. This document is one of the results of the first twelve months of activities that includes the definition of policies stemming from the threats and attacks that are relevant for AI applications as well as security tools we developed to harden the architecture of AI-SPRINT with regards to security.

This document has presented the security tools of the AI-SPRINT architecture such as the SCONE runtime, the cross compiler, the sconify image tool, the CAS (configuration and attestation service). For each of these components the document provides details on their functionality as well as where these components are in place referring to the federated learning application as a running example. The document also presented the efforts for secure network communications that cover edge communication as well as communication within cloud environments through transparent encryption provided by the SCONE runtime.

The second version of this document will be provided in M24 in D4.2 “Second release and evaluation of the security tools”, while the final version will be provided in M30 in D4.3 “Final release and evaluation of the security tools”, after completing the different phases of developments that will include the feedback of the use cases and other work packages.

References

- [1] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, L. Zhang. 2016. Deep Learning with Differential Privacy. In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS).
- [2] I. Ahmad, T. Kumar, M. Liyanage, J. Okwuibe, M. Ylianttila, A. Gurtov, "Overview of 5G Security Challenges and Solutions," in IEEE Communications Standards Magazine, vol. 2, no. 1, pp. 36-43, MARCH 2018, doi: 10.1109/MCOMSTD.2018.1700063.
- [3] S. Arnautov, B. Trach, F. Gregor, T. Knauth, A. Martin, C. Priebe, J. Lind, D. Muthukumaran, D. O’Keeffe, M. L. Stillwell, D. Goltzsche, D. Eysers, R. Kapitza, P. Pietzuch, C. Fetzer. 2016. SCONE: Secure Linux Containers with Intel SGX. In 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI).
- [4] M. Bailleu, D. Giantsidi, V. Gavrielatos, D. Le Quoc, V. Nagarajan, P. Bhatotia. 2021. Avocado: A Secure In-Memory Distributed Storage System. In 2021 USENIX Annual Technical Conference.
- [5] A. N. Bhagoji, S. Chakraborty, P. Mittal, S. Calo. 2019. Analyzing federated learning through an adversarial lens. In International Conference on Machine Learning. PMLR.
- [6] P. Blanchard, E. Mahdi, E. Mhamdi, R. Guerraoui, J. Stainer. 2017. Machine learning with adversaries: Byzantine tolerant gradient descent. In Proceedings of the 31st International Conference on Neural Information Processing Systems. 118–128.
- [7] X. Cao, J. Jia, N. Z. Gong. 2021. Provably Secure Federated Learning against Malicious Clients. In Proceedings of the AAAI Conference on Artificial Intelligence.
- [8] V. Costan, S. Devadas. 2016. Intel SGX Explained. IACR Cryptol. ePrint Arch.
- [8] M. Fang, X. Cao, J. Jia, N. Gong. 2020. Local model poisoning attacks to byzantine-robust federated learning. In 29th USENIX Security Symposium.
- [9] James C Gordon. [n.d.]. Microsoft Azure Confidential Computing with Intel SGX. <https://software.intel.com/content/www/us/en/develop/blogs/microsoft-azure-confidential-computing-with-intel-sgx.html>. Accessed: Sept 2021.
- [10] F. Gregor, W. Ozga, S. Vaucher, R. Pires, S. Arnautov, A. Martin, V. Schiavoni, P. Felber, C. Fetzer, et al. 2020. 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).
- [11] P. Karnati. 2018. Data-in-use protection on IBM Cloud using Intel SGX. <https://www.ibm.com/cloud/blog/data-use-protection-ibm-cloud-using-intel-sgx>. Accessed: Sept 2021.
- [12] J. Konečný, H. B. McMahan, F. X Yu, P. Richtárik, A. T. Suresh, D. Bacon. 2016. Federated learning: Strategies for improving communication efficiency. arXiv preprint arXiv:1610.05492.
- [13] R. Krahn, D. Dragoti, F. Gregor, D. Le Quoc, V. Schiavoni, P. Felber, C. Souza, A. Brito, C. Fetzer. 2020. TEEMon: A continuous performance monitoring framework for TEEs. In Proceedings of the 21th International Middleware Conference (Middleware).
- [14] R. Kunkel, D. Le Quoc, F. Gregor, S. Arnautov, P. Bhatotia, C. Fetzer. 2019. Tensorscone: A secure tensorflow framework using Intel SGX. arXiv preprint arXiv:1902.
- [15] D. Le Quoc, F. Gregor, S. Arnautov, R. Kunkeland, P. Bhatotia, C. Fetzer. 2020. secureTF: A Secure Tensor-Flow Framework. In Proceedings of the 21th International Middleware Conference (Middleware).

- [16] D. Le Quoc, F. Gregor, J. Singh, C. Fetzer. 2019. SGX-PySpark: Secure Distributed Data Analytics. In Proceedings of the World Wide Web Conference (WWW).
- [17] A. Martin, C. Lian, F. Gregor, R. Krahn, V. Schiavoni, P. Felber, C. Fetzer. 2021. ADAM-CS: Advanced Asynchronous Monotonic Counter Service. In 2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN).
- [18] V. Mugunthan, A. Peraire-Bueno, L. Kagal. 2020. PrivacyFL: A simulator for privacy-preserving and secure federated learning. In Proceedings of the 29th ACM International Conference on Information & Knowledge Management (CIKM).
- [19] W. Ozga, C. Fetzer, et al. 2021. Perun: Confidential Multistakeholder Machine Learning Framework with Hardware Acceleration Support. In IFIP Annual Conference on Data and Applications Security and Privacy. Springer, 189–208.
- [20] W. Ozga, D. Le Quoc, C. Fetzer. 2020. A practical approach for updating an integrity-enforced operating system. In Proceedings of the 21th International Middleware Conference (Middleware).
- [21] W. Ozga, D. Le Quoc, C. Fetzer. 2021. Perun: Secure Multi-Stakeholder Machine Learning Framework with GPU Support. arXiv preprint arXiv:2103.16898 (2021).
- [22] W. Ozga, D. Le Quoc, C. Fetzer. 2021. WELES: Policy-driven Runtime Integrity Enforcement of Virtual Machines. arXiv preprint arXiv:2104.14862 (2021).
- [23] G. A. Reina, A. Gruzdev, P. Foley, O. Perepelkina, M. Sharma, I. Davidyuk, I. Trushkin, M. Radionov, A. Mokrov, D. Agapov, J. Martin, B. Edwards, M. J. Sheller, S. Pati, P. N. Moorthy, S. Wang, P. Shah, S. Bakas. 2021. OpenFL: An open-source framework for Federated Learning. arXiv:2105.06413
- [24] V. Scarlata, S. Johnson, J. Beaney, P. Zmijewski. 2018. Supporting third party attestation for Intel SGX with Intel data center attestation primitives. White paper.
- [25] J. Singh, J. Cobbe, D. Le Quoc, Z. Tarkhani. 2020. Enclaves in the Clouds: Legal considerations and broader implications. Queue (2020).
- [26] J. Singh, J. Cobbe, D. Le Quoc, Z. Tarkhani. 2021. Enclaves in the Clouds. Commun. ACM 64, 5 (2021), 42–51.
- [27] C.-C. Tsai, D. E. Porter, M. Vij. 2017. Graphene-SGX: A Practical Library OS for Unmodified Applications on SGX. In USENIX Annual Technical Conference (USENIXATC 17).
- [28] P. Voigt, A. Von dem Bussche. 2017. The EU general data protection regulation (GDPR). A Practical Guide, 1st Ed., Cham: Springer International Publishing (2017).
- [29] C. Xie, O. Koyejo, I. Gupta. 2020. Fall of empires: Breaking Byzantine-tolerant SGD by inner product manipulation. In Uncertainty in Artificial Intelligence. PMLR.
- [30] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.
- [31] LeCun and C. Cortes. MNIST handwritten digit database. 2010.
- [32] 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, 2016.
- [33] B. Xu, N. Wang, T. Chen, and M. Li. Empirical evaluation of rectified activations in convolutional network. arXiv preprint arXiv:1505.00853, 2015.
- [34] C.-C. Tsai, D. E. Porter, and M. Vij. Graphene-SGX: A practical library OS for unmodified applications on SGX. In Proceedings of the USENIX Annual Technical Conference (USENIX ATC), 2017.

- [35] Baumann, M. Peinado, and G. Hunt. Shielding Applications from an Untrusted Cloud with Haven. In Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI), 2014.
- [36] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In Proceedings of the 31th AAAI Conference on Artificial Intelligence (AAAI), 2017.
- [37] 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, 2016.
- [38] 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, 2017
- [39] D. Moro, M. Peuster, H. Karl, and A. Capone, “FOP4: Function offloading prototyping in heterogeneous and programmable network scenarios,” in 2019 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN). IEEE, 2019, pp. 1–6.