Grant Agreement N°: 101020259
Topic: SU-DS02-2020

ARCADIAN-IoT

Autonomous Trust, Security and Privacy Management Framework for IoT

D3.3: Horizontal Planes – final version

Revision: v1.0

Work package	3
Task	Tasks 3.1, 3.2, 3.3, 3.4, and 3.5
Due date	31/01/2024
Submission date	05/02/2024
Deliverable lead	ATOS
Version	1.0
Partner(s) / Author(s)	RISE: Alfonso Iacovazzi, Han Wang IPN: Paulo Silva, Vitalina Holubenko, José Gaspar, Rúben

	Leal, Sérgio Figueiredo UWS: Jose M. Alcaraz Calero, Qi Wang, Antonio Matencio Escolar, Ignacio Sánchez Navarro, Pablo Benlloch Caballero 1GLOBAL: João Casal, Afonso Paredes, Ivo Vilas Boas XLAB: Tilen Marc, Nejc Bat MARTEL: Giacomo Inches, Gabriele Cerfoglio, Andrea Falconi, Valerio Cantore BOX2M: Alexandru Gliga, Ovidiu Diaconescu, Marian Macoveanu ATOS: Ross Little, Sergio Sanz

Abstract

This public technical report constitutes the deliverable D3.3 of ARCADIAN-IoT, a Horizon2020 project with **the grant agreement number 101020259**, under the topic **SU-DS02-2020**. D3.3 is the third and final submission of a series deliverables to detail the findings, achievements, and outcomes resulted from WP3 research activity. The deliverable builds on the previous submission D3.2 to include the final results of WP3.

WP3 is dedicated to the technological development of the components in the Horizontal Planes of ARCADIAN-IoT framework for each use case defined in Task 2.1 (Use cases specification and planning) and before being integrated in WP5. WP3 is organized in five main tasks (from Task 3.1 to 3.5), each of which focusing on the definition and development of one or more components in the Horizontal Planes.

Keywords: ARCADIAN-IoT, Privacy preservation, Intrusion detection, Intrusion prevention, Self-healing, Self-protection, Hardened Encryption, Permissioned Blockchain, Cyber Threat Intelligence.

Document Revision History

Version	Date	Description of change	List of contributor(s)
V0.1	08/05/2023	Preliminary Template	RISE
V0.2	05/12/2023	Updates to general sections by ATOS and all component sections by relevant partners	ATOS, RISE, IPN, UWS, 1GLOBAL, BOX2M, MARTEL, XLAB
V0.3	08/12/2023	Internal review	RISE
V0.4	31/04/2024	Updates after internal review	ATOS, RISE, IPN, UWS, 1GLOBAL, BOX2M, MARTEL, XLAB
V1.0	02/02/2024	Final quality check	IPN, ATOS

Project co-funded by the European Commission under SU-DS02-2020		
Nature of the deliverable:	OTHER	
Dissemination Level		

PU	Public, fully open, e.g. web	√
CI	Classified, information as referred to in Commission Decision 2001/844/EC	
CO	Confidential to ARCADIAN-IoT project and Commission Services	

** R: Document, report (excluding the periodic and final reports)*
DEM: Demonstrator, pilot, prototype, plan designs
DEC: Websites, patents filing, press & media actions, videos, etc.
OTHER: Software, technical diagram, etc

EXECUTIVE SUMMARY

ARCADIAN-IoT Deliverable 3.3 is the final WP3 technical report presenting the complete research findings and outcomes of activities performed for each component within the Horizontal Planes of the ARCADIAN-IoT framework (comprising Privacy, Security, and Common planes). This deliverable is an update to the previous Deliverable 3.2, which presented the interim results focused on the 1st prototype (P1). D3.3 stands as self-contained document, providing a comprehensive report on the development of each WP3 Horizontal Plane component, without the need to refer to previous deliverables. Each component is presented in a standardized format: the presentation starts with an overview and a recall of the corresponding objectives and target Key Performance Indicators (KPIs), followed by a description of the technology research, encompassing the research findings and produced outcomes. Additionally, design specifications are given, helping to understand the (internal) technical details of each component, as well as the interfaces with other components and organizations making use of ARCADIAN-IoT framework. Finally, the results of evaluations along with plans for future work beyond the project's completion are described for each component.

The work performed in WP3, and described in this document, is organized in five main tasks (from Task 3.1 to 3.5), each of which focuses on the definition and development of one or more components in the Horizontal Planes. **Task 3.1** aims at creating an efficient **Permissioned Blockchain** based on the open-source alternative Hyperledger Fabric, that will in turn support other ARCADIAN-IoT components such as Decentralized Identifiers and Reputation components. **Task 3.2** focuses on the development of **Hardened Encryption** mechanisms to protect and authenticate private data in IoT devices. **Task 3.3** is devoted to creating **privacy preserving** technologies: (i) a dependable and privacy preserving classifier based on **Federated AI** algorithms which will also guarantee the source and data integrity, and (ii) a **Self-aware Data Privacy** component that will enhance the way data privacy is managed in IoT contexts. A novel IoT-specific **Cyber Threat Intelligence** system based on the MISP (Malware Information Sharing Platform) toolset which will provide privacy preserving data sharing capability and IoT-specific Indicators of Compromises is being developed in **Task 3.4**. Finally, **Task 3.5** aims to define and implement the monitoring and recovering functionalities which will be provided by the following components: (i) a **Network flow monitoring** agent, enhancement of existing Network Intrusion Detection Systems able to detect known malicious Distributed Denial of Service (DDoS) along the entire IoT infrastructure, (ii) a **Device Behaviour Monitoring** component able to detect anomalous behaviours that occurs on IoT devices, (iii) a **Network Self-healing**, (iv) **IoT Network Self-protection**, and (v) **IoT Device Self-protection** capabilities to mitigate and recover from cyberattacks against IoT networks.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

3PP	3rd Party Provider
4G	4th Generation
5G	5th Generation
5GS	5G System
aiotID	ARCADIAN-IoT Identity
ABE	Attribute Based Encryption
ADFA-LD	Australian Defence Force Academy Linux Dataset
AI	Artificial Intelligence
AIDS	Anomaly IDS
API	Application Programming Interface
CP-ABE	Ciphertext-Policy ABE
CSIRT	Computer Security Incident Response Team
CTI	Cyber Threat Intelligence
dApp	decentralized Application
DCSP	Data Centre Service Provider
DDoS	Distributed Denial of Service
DID	Decentralized Identifier
DLT	Distributed Ledger Technology
DSC	Datapath Security Controller
xSIM	Any SIM form (e.g. SIM, eSIM, iSIM)
eUICC	embedded Universal Integrated Circuit Card
FE	Functional Encryption
FL	Federated Learning
FTP	File Transfer Protocol
GENEVE	Generic Network Virtualization Encapsulation
GRE	Generic Routing Encapsulation
GSMA	Global System for Mobile Communications Association
GSMA-SAS	GSMA's Security Accreditation Scheme
GTP	GPRS (General Packet Radio Service) Tunnelling Protocol
GUI	Graphical User Interface
HE	Hardened Encryption
HIDS	Host-based IDS
HTTP	Hypertext Transfer Protocol
ID	Identifier
IDS	Intrusion Detection System
IID	Independent and Identically Distributed
IMSI	International Mobile Subscriber Identity
IoC	Indicator of Compromise
IoT	Internet of Things
IOTA	Internet of Things Application

IoT SAFE	IoT SIM Applet For Secure End-2-End Communication
IP	Internet Protocol
IPR	Intellectual Property Rights
IPS	Intrusion Prevention System
IT	Information Technology
JSON	JavaScript Object Notation
KPI	Key Performance Indicator
LTE	Long-Term Evolution
M2M	Machine to Machine
MAC	Media access control
MISP	Malware Information Sharing Platform
ML	Machine Learning
MPLS	Multiprotocol Label Switching
MVNO	Mobile Virtual Network Operator
NAA	Network Access Application
NB-IoT	Narrowband IoT
NBI	North Bound Interface
NFM	Network Flow Monitoring
NIDS	Network-based IDS
OPA	Open Policy Agent
OS	Operating System
OVS	OpenVSwitch
PAP	Policy Administration Point
PCA	Protection Control Agent
PD	Protection Decider
PDP	Policy Decision Point
PEP	Policy Enforcement Point
PoA	Proof of Authority
PPP	Public Private Partnership
REST	Representational State Transfer
RIA	Resource Inventory Agent
RoT	Root of Trust
RPC	Remote Procedure Call
SADP	Self-Aware Data Privacy
SDN	Software Defined Network
SFMA	Security Flow Monitoring Agent
SHA	Secure Hash Algorithm
SHDM	Self-Healing Decision Manager
SIDS	Signature IDS
SIM	Subscriber Identity Module
SMOTE	Synthetic Minority Oversampling Technique
SP	Service Provider

SSH	Secure Shell
SSI	Self-Sovereign Identity
STIX	Structured Threat Information eXpression
SVM	Support Vector Machine
TAXII	Trusted Automated eXchange of Intelligence Information
TCP	Transmission Control Protocol
TEID	Tunnel Endpoint Identification
TLS	Transport Layer Security
TPR	True Positive Rate
UDP	User Datagram Protocol
UICC	Universal Integrated Circuit Card
URL	Uniform Resource Locator
VISP	Virtualisation Infrastructure Service Provider
VLAN	Virtual Local Area Network
VxLAN	Virtual Extensible Local Area Network
WAC	Web Access Control
WP	Work Package
XDP	eXpress Data Path
XML	eXtensible Markup Language

1 INTRODUCTION

1.1 ARCADIAN-IoT and its Horizontal Planes

The ARCADIAN-IoT project aims to develop a cyber security framework relying on a novel approach to manage and coordinate, in an integrated way, identity, trust, privacy, security, and recovery in IoT systems. The proposed approach organizes the multiple cyber security functionalities offered by the framework into several planes combined in an optimized way to support the end-to-end services. In particular, the framework includes three Vertical Planes devoted to identity, trust, and recovery management, and three Horizontal Planes supporting the Vertical Planes by managing privacy of data, monitoring security of entities, and providing Permissioned Blockchain and Hardened Encryption technologies (see Figure 1).

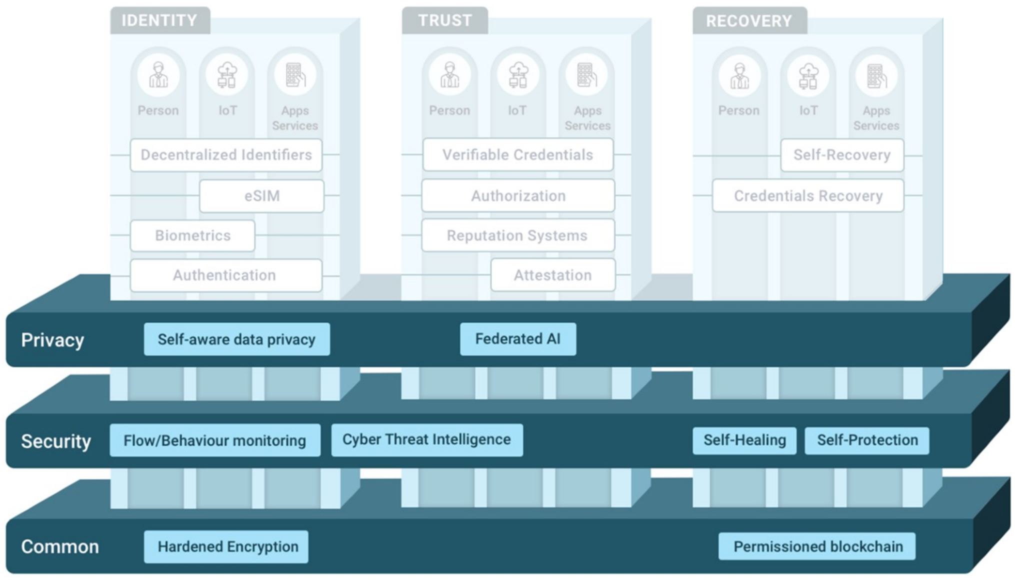


Figure 1 - ARCADIAN-IoT Conceptual representation highlighting Horizontal Planes

The Horizontal Planes of the ARCADIAN-IoT framework (consisting of ten main components) are organized as follows:

- The **Privacy Plane**, which aims to provide functionalities for the privacy-preserving management of confidential or sensitive data involving persons' entities, includes the (i) Self-aware Data Privacy and (ii) Federated Artificial Intelligence (Federated AI) components.
- The **Security Plane** contains all the cyber security features required for the monitoring, prevention, management, and recovery; it comprises the (i) Network Flow Monitoring, (ii) Behaviour Monitoring, (iii) Cyber Threat Intelligence, (iv) Network Self-protection, (v) IoT Device Self-protection, and (vi) Network Self-healing components.
- The **Common Plane** includes the two components that provide common functionalities to the Vertical Planes, i.e., (i) the Hardened Encryption[1] and (ii) Permissioned Blockchain.

Table 1 shows how the components in the Horizontal Planes are grouped per plane and in which task of the project are developed.

[1] Hardened Encryption comes in 2 variants: a) SIM as RoT b) cryptochip as RoT

Plane	Component	Task
Privacy Plane	Self-aware Data Privacy	3.3
	Federated AI	3.3
Security Plane	Network Flow Monitoring	3.5
	Behaviour Monitoring	3.5
	Cyber Threat Intelligence	3.4
	Network Self-protection	3.5
	IoT Device Self-protection	3.5
	Network Self-healing	3.5
Common Plane	Hardened Encryption[1]	3.2
	Permissioned Blockchain	3.1

Table 1 - Mapping the components to WP tasks.

This deliverable is organized in three main sections mapping the three Horizontal Planes. Every section is split into multiple subsections, each of which describing the research findings for a specific component in the plane. The description of a component reports:

1. An overview of the developed component, together with a recall of related requirements, objectives, and Key Performance Indicators (KPIs).
2. A report of the technology research activities: giving an overview of the technology used, summarizing the research findings, achievements and describing the produced resources, including the references to the source code that is shared by the partners, whenever possible.
3. The design specification of the component, detailing the logical architecture, interfaces to other components, APIs for using the component, and other technical details.
4. The evaluation results of the component.
5. The plan for future work beyond the project's completion.

This deliverable supersedes all previous WP3 deliverables and serves as a self-contained deliverable describing the research, design and overall results for each of the ARCADIAN-IoT components within the Horizontal Planes.

1.2 Objectives

WP3 aims at contributing to the achievements of six main objectives in the ARCADIAN-IoT project, defined in the ARCADIAN-IoT grant agreement. Each objective comes with individual Key Performance Indicators (KPIs), also defined in the grant agreement. Besides these initial KPIs, the component-specific KPIs have been revised to provide accurate and measurable indicators of the success of the project. The summary of the project's objectives and WP3's contribution to the associated KPIs are listed here, along with additional details of the WP3 components.

- **To create a decentralized framework for IoT systems - ARCADIAN-IoT framework and enable distributed security and trust in management of persons' identification.**
    - Use the permissioned blockchain to publish trusted information to third parties in the ARCADIAN-IoT framework.
    - Deployment of permissioned blockchain in IoT environments.
    - Train partners to deploy a blockchain network
- **Provide distributed and autonomous models for trust, security, and privacy –**

enablers of a Chain of Trust.

  - Improve communication efficiency in federated learning.
  - Improve robustness in federated learning against model and data poisoning attacks.
  - Provide a new data rebalancing technique for federated learning setting outperforming state-of-the-art methods (K-SMOTE, SMOTE, up/down-sampling) in term of accuracy and efficiency.
  - Continuous training of IDS models in IoT devices.
  - Ability to process different input types with non-root privileges (e.g., syscalls, auth, rep).
  - Deployment in heterogenous devices.

- **Provide a Hardened Encryption with recovery ability.**
  - Encryption library enabling fine-grained access control over data with APIs in at least 4 programming languages and demonstrate its use on at least 2 types of devices/platforms.
  - Add Root of Trust information to encrypted data with xSIM[2]-based digital signatures.
  - Provide secure and scalable key management and delegation synchronized with the decentralized identity management, multi-factor authentication and self-recovery.
  - Provide efficient implementation of the process with encryption times comparable to the state-of-the-art results on multiple devices, and with RoT signatures in acceptable time for the communication processes.
  - Provide secure and scalable key management, in both device & middleware sides, covering all operations lifecycle situations (creation, fulfilment, regular data communications, removal, recovery) for devices fleets.
  - Provide efficient implementation of the process at a device and the middleware side in acceptable time for the communication (telecommunication protocols and upper layer OSI protocols wise) processes.
- **Self and coordinated healing with reduced human intervention.**
  - Provide new alert metadata information about IoT and 5G flow structure, within the supported network segments with a number of >= 4 (Edge, Core, RAN and Transport).
  - Provide transversal detection capabilities to protect, simultaneously tenant infrastructure and the infrastructure provider by supporting >=4 encapsulations and tunnelling protocols widely used in overlay networks inherent to 4G/5G IoT mobile infrastructures such as VXLAN, GRE, GENEVE and GTP.
  - Novel traffic classifier able to deal with data paths in 4G/5G IoT networks (overlay traffic with several levels of nested encapsulation).
  - OpenFlow protocol extension to provide a flexible and extended programmability of the data plane.
  - OVS Netlink API extension for inter-process communication kernel-user space.

[2] xSIM stands for any form of Subscriber Identity Modules, like SIM, eSIM or iSIM

  - Policy enforcement based on ensemble of risk levels, indicators of compromise, reputation and threat information.
  - Autonomous self-protection mechanism for heterogeneous operating systems and devices.
  - Provide self-healing for overlay networks and IoT networks supporting >= 4 encapsulation and tunnelling protocols widely used in overlay networks inherent to 4G/5G IoT mobile infrastructures such as VXLAN, GRE, GENEVE or GTP.
  - Provide dynamic and distributed enforcing of protection/healing policies in 7 or more network segments (Edge, Core, RAN, Transport, Backbone, Enterprise, Backhaul, Midhaul) in 20 seconds or less for up to 4096 IoT devices sending a combined bandwidth of 10 Gbps of malicious traffic; or 2048 IoT devices sending a combined bandwidth of 25 Gbps of malicious traffic.
  - Provide network topology understanding to reduce human intervention (towards 0% of human intervention) supporting the coordination of recovery to pre-defined trust levels (>= 95% of flow services prior to anomalous behaviour) using such topology.
- **Enable proactive information sharing for trustable Cyber Threat Intelligence and IoT Security Observatory.**
  - Enable the aggregation and processing of IoCs generated by internal or external sources.
  - Provide ML models for automated IoC clustering.
  - Provide ML models for automated IoC ranking.
  - Provide ML models for automated Event/Galaxy classification.

1.3 Component description methodology

The work associated to each ARCADIAN-IoT component is described in the upcoming sections according to the following structure:

- First, the **Overview** provides the overall research and development scope associated to the component, including requirements, innovation-driven objectives and associated KPIs.
- The **Technology Research** spans both background relevant to the work performed in ARCADIAN-IoT (e.g. prior related solutions or knowledge), key research findings and achievements attained during the project, as well as resources produced as outcomes from the research and development work (e.g. datasets or software code).
- **Design specification** describes the component from a software design perspective, presenting relevant specifications such as logical or deployment architectures, internal and external interfaces, or other technical specifications specific to the scope of the research area.
- In **Evaluation and results**, performed experiments and associated evaluation results are described.
- Finally, in **Future Work**, the research and improvement opportunities identified as a result of ARCADIAN-IoT work are presented.

2 PRIVACY PLANE

2.1 Self-aware Data Privacy

2.1.1 Overview

2.1.1.1 Description

Within the scope of the ARCADIAN-IoT project, Martel developed a component to empower the users to better control privacy of their data, in particular by allowing the definition of user- defined privacy policies for data, and by suggesting policies specified on similar data. The Self-Aware Data Privacy (SADP) component includes two main modules: a policy management module which enforces data privacy via the definition of privacy policies – in the context of this component, policies relate specifically to attributed-based data encryption and/or anonymization -, and a recommender module which, by assessing policy similarity, suggests privacy policies. The policy management module leverages a background of Martel to ensure that privacy policies and data access are consistent.

2.1.1.2 Requirements

A recall of the high-level requirements that have been previously defined and provided in deliverable D2.4 is given below.

Requirement 2.1.1 – User defined policies: An authorized user can access the system and specify for a given data source or data property the security policies that allows to protect the data either when entering into the system or exiting – or both.

Requirement 2.1.2 – Policies validation: Security policies can be specified in a machine-readable format (e.g., JSON) and validated against a schema interpreter to assess their validity and applicability.

Requirement 2.1.3 – Data secured: Data sources or data attribute scan be secured by a given methodology: anonymisation, pseudo-anonymisation, encryption, and advanced encryption (hardened) based on attribute-based encryption.

Requirement 2.1.4 – Recommender: The system is able to recognise similarity between new data and existing data by key, attributes and/or semantic; This result is eventually displayed to users for facilitating the issuing of security policies.

2.1.1.3 Objectives and KPIs

The main objective of the component is to empower users to better control privacy over their data, in particular by allowing the definition of user-defined privacy policies for specific attributes and by suggesting policies specified on similar data. The following KPIs allowed to verify the achievement of these objectives.

KPI scope	
Making sure the protection algorithm is flexible and can adapt to different needs	
Measurable Indicator	
Number of policies handled by the protection algorithm	
Target value	***Achieved value***
5 encryption/anonymisation kind of policies handled	1 anonymisation policy implemented directly, 1 encryption algorithm supported via the Integration with Hardened Encryption, 10+ more algorithm schemes supported by the

	integrated library[3]

KPI scope	
Verify that the recommender system can retrieve similar policies and evaluate its effectiveness	
Measurable Indicator	
Precision/recall combined in the F1 score for the recommender algorithms	
Target value	**Achieved value**
≥80%	Approx. 81.3%

KPI scope	
Making sure the component is usable and can simplify the issuing of policies by the users	
Measurable Indicator	
Usability of the policing issuer (Likert scale).	
Target value	**Achieved value**
90% Positive Feedback	80% Positive Feedback (lessons learned: the interface implementation and testing required more effort than foreseen)

2.1.2 Technology research

2.1.2.1 Background

The investigation conducted within the framework of the project had an applicative focus on state-of-the-art algorithms for the efficient implementation of policy management and enforcement.

The Self-aware data privacy component leverages two state-of-the-art tools:

Open Policy Agent (OPA[4]), an open-source, general-purpose policy engine which decouples policy decision-making from policy enforcement, generating policy decisions by evaluating a query input (can be any structured data, e.g., JSON) against policies and data.

Envoy Proxy[5], an open-source edge and service proxy designed for single services and applications, as well as a communication bus and "universal data plane" designed for large microservice "service mesh" architectures.

In addition to those tools, in the Recommender component we leverage NLTK[6] to assess similarity between attributes of data objects.

2.1.2.2 Research findings and achievements

The two background tools were combined and concatenated in two sub-modules: Anubis and Amon which together constitute the policy management module of the Self-aware data privacy. The two sub-components are enriched with a third one: the Recommender sub-component. These three constitute the main achievement for Self-aware Data Privacy within WP3 and within the framework of the project.

[3] https://github.com/fentec-project/gofe
[4] https://www.openpolicyagent.org
[5] https://www.envoyproxy.io/docs/envoy/latest
[6] https://www.nltk.org/

Anubis[7] is a flexible policy enforcement solution that makes easier to reuse security (access) policies across different services entailing the same data. It is background developed by Martel within the Cascade Funding mechanisms of the EC project DAPSI (GA871498) and adapted in ARCADIAN-IoT to provide access polices for Amon, which is one of the main project's foreground and allows to: 1) define anonymization and encryption policies for data and 2) apply encryption (and decryption policies) on data.

The other major project foreground is the Recommender implementation with its GUI, which make easier to specify protection policies, even for non-technical users.

2.1.2.3 Produced resources

In Figure 2 we recall the logical architecture initially included in D3.1 and provide an update of the status of the implementation, with reference to the technologies and components presented in the previous sections.

All bullets 1 to 5 are now completed, including the finalisation of the Amon sub-module (4) and the recommender system and GUI implementation, which were pending in the previous release of this deliverable (D3.2).

(1) Within ARCADIAN-IoT, one of the first activities was to adapt the Access Management sub-modules (Anubis) to serve the purpose of the Self-aware data privacy.

We then implemented Amon as follow:

(2) A Policy Decision Point (PDP) Based on the open source, general-purpose policy engine Open Policy Agent (OPA), allowing the decoupling between decision-making vs. policy enforcement vs. user access (Anubis).

(3) Policy Administration Point (PAP), a custom API (Open-source) to implement the "Policy Description" component follow the Data Model originally defined for user, data, access and security algorithm.

The (4) Policy Enforcement Point (PEP) is the combination of the API developed as PAP (2) and the Envoy plugin of OPA and will allow to anonymize/encrypt the data, eventually leveraging the encryption/decryption (Go) Libraries or API provided by XLAB

Finally, the (5) Recommender design and implementation as well as the GUI one, for specifying encryption/anonymization policies. The GUI leveraged and extended the one already defined for Anubis and illustrated below, to connect with Amon as well.

The 3 sub-components are available in the respective repositories[8] as indicated here below:

1. Anubis: https://github.com/orchestracities/anubis
2. Amon: https://github.com/orchestracities/amon
3. Recommender: https://github.com/orchestracities/recommender-ui

[7] https://github.com/orchestracities/anubis

[8] Amon and Recommender are not yet publicly accessible. Explicit access request can be addressed to Martel (lab@martel-innovate.com)

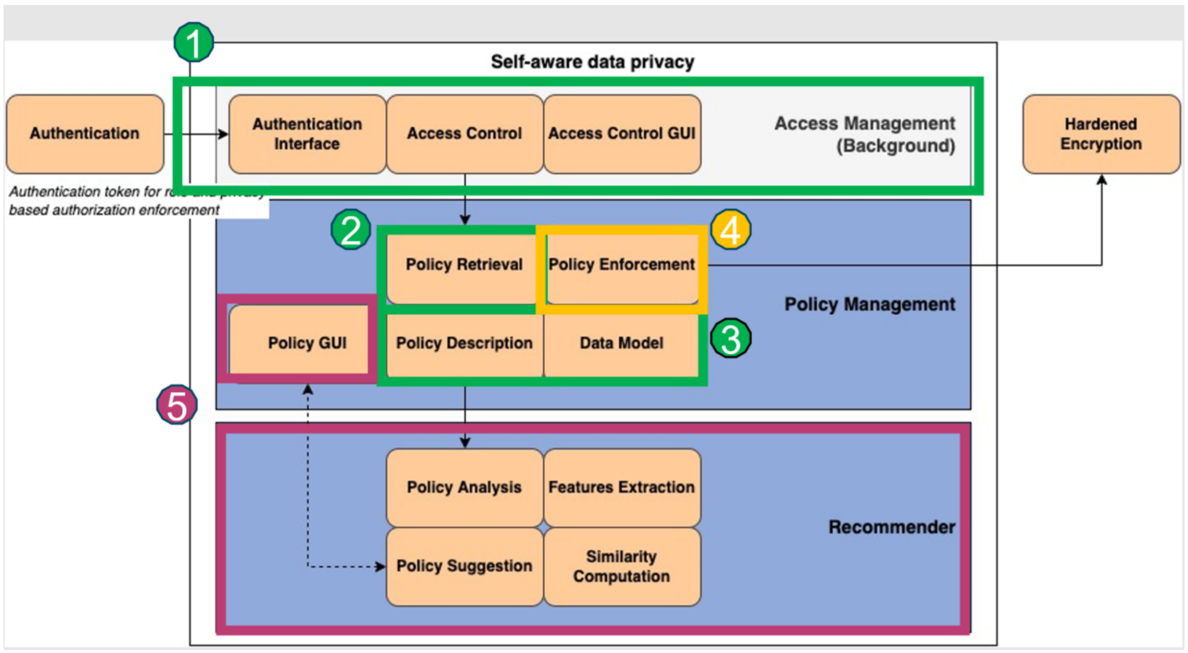


Figure 2 - Logical architecture for the Self-Aware Data Privacy component

2.1.3 Design specification

In Figure 3 we illustrated the two sub-modules Anubis and Amon in relation with the logical architecture presented above and provide a better inside on the interaction of the different components. This is described in the next sections into more details.

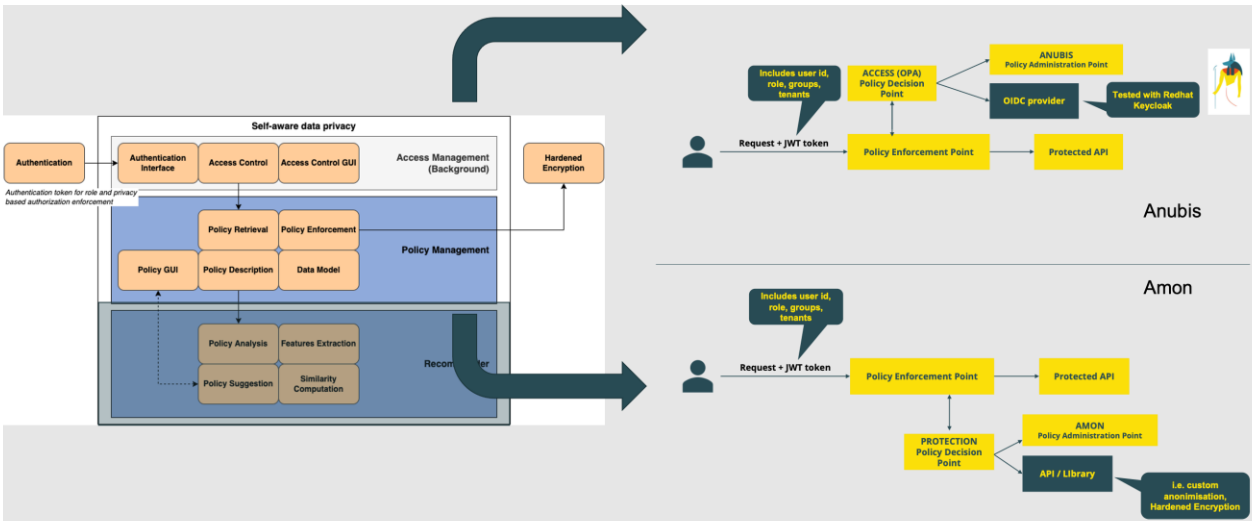


Figure 3 - Self-Aware Data Privacy logical architecture

2.1.3.1 Logical architecture view

We first provide a first overview of the Anubis component, then describe Amon and their mutual interaction.

In term of policy enforcement, Anubis adopts a standard architecture: a client request for a resource to an API, and based on the defined policies, the client is able or not to access the resource. Figure 4 shows the current architecture.

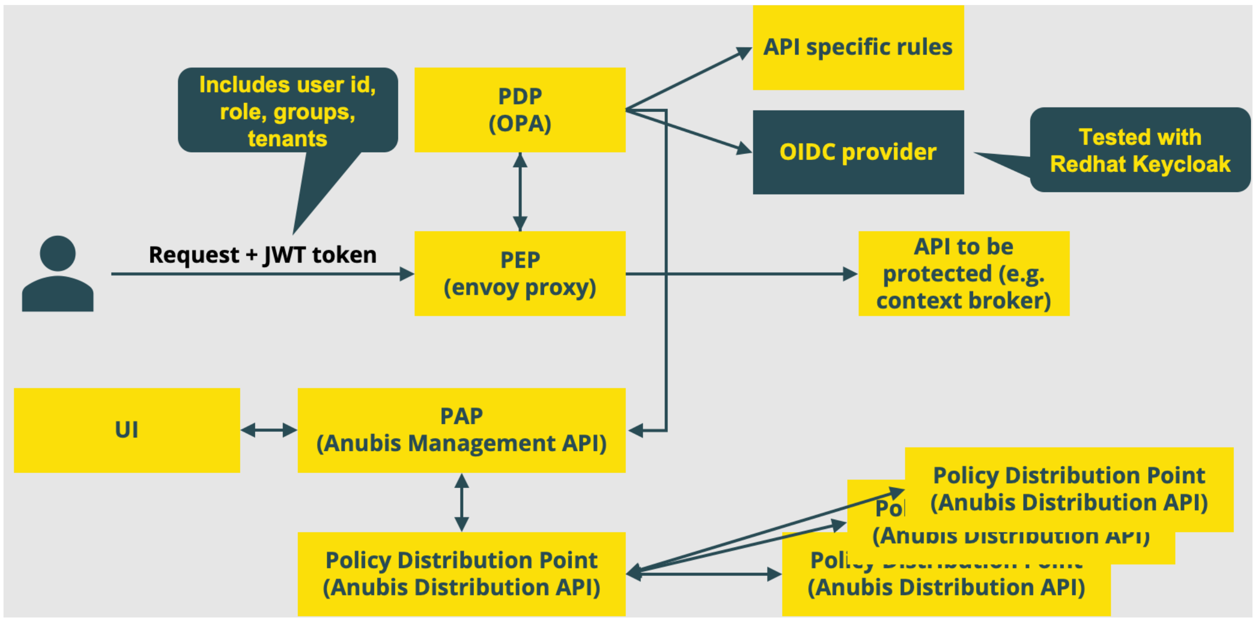

API specific rules
Includes user id, role, groups, tenants
PDP (OPA)
OIDC provider
Tested with Redhat Keycloak
Request + JWT token
PEP (envoy proxy)
API to be protected (e.g. context broker)
UI
PAP (Anubis Management API)
Policy Distribution Point (Anubis Distribution API)
Policy Distribution Point (Anubis Distribution API)
Policy Distribution Point (Anubis Distribution API)
Policy Distribution Point (Anubis Distribution API)

Figure 4 - Anubis logical architecture

Anubis currently supports only Role Based Access Control policies. Policies are stored in the policy management API, that supports the translation to Web Access Control (WAC) policies and to a data input format supported by OPA, the engine that performs the policy evaluation.

At the time being, the API specific rules have been developed specifically for the NGSIv2 Context Broker[9], Anubis management, and JWT-based authentication. In addition to that, thanks to the libp2p middleware[10], polices can also be distributed across different Policies Administration Points. The architecture decouples the PAP from the distribution middleware as in the picture above.

On the other hand, Amon aims to be a data protection policy enforcement middleware, i.e. it allows to define anonymization and encryption policies for data on the one side and on the other to apply encryption (and decryption policies) on data. Amon complements Anubis, that retains control over access to data (and access to decrypted data). The three main properties of Amon are: 1) Define policies for data anonymisation and encryption 2) Decouple the policies from the application of the encryption and anonymisation techniques 3) Is Attribute based.

In term of data protection policy enforcement, Amon leverages a similar architecture to Anubis: a client request for a resource to an API, and based on the defined policies, the client receives back the resource with part of the data protected (e.g. encrypted) or not. Figure 5 shows Amon's logic architecture.

[9] https://fiware-orion.readthedocs.io/en/master/
[10] https://libp2p.io/

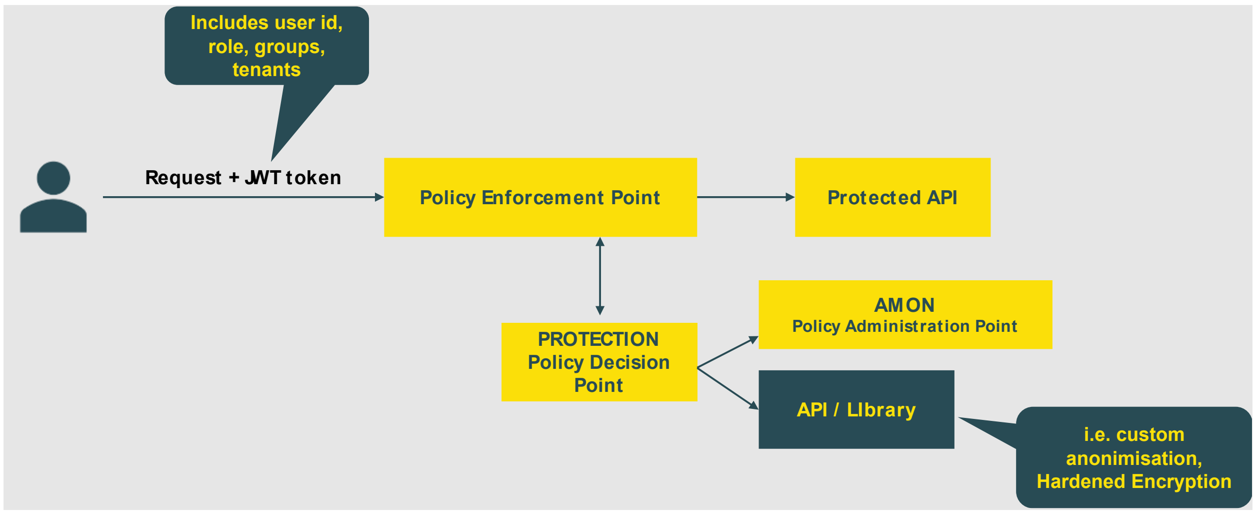

Includes user id, role, groups, tenants
Request + JWT token
Policy Enforcement Point
Protected API
AMON
Policy Administration Point
PROTECTION
Policy Decision Point
API / Library
i.e. custom anonimisation, Hardened Encryption

Figure 5 - Amon logical architecture

In Figure 6 we illustrate the joint action Anubis-Amon, in particular by the PEP which is able to concatenate the chain of authorization first for access (Anubis), then for data protection (Amon).

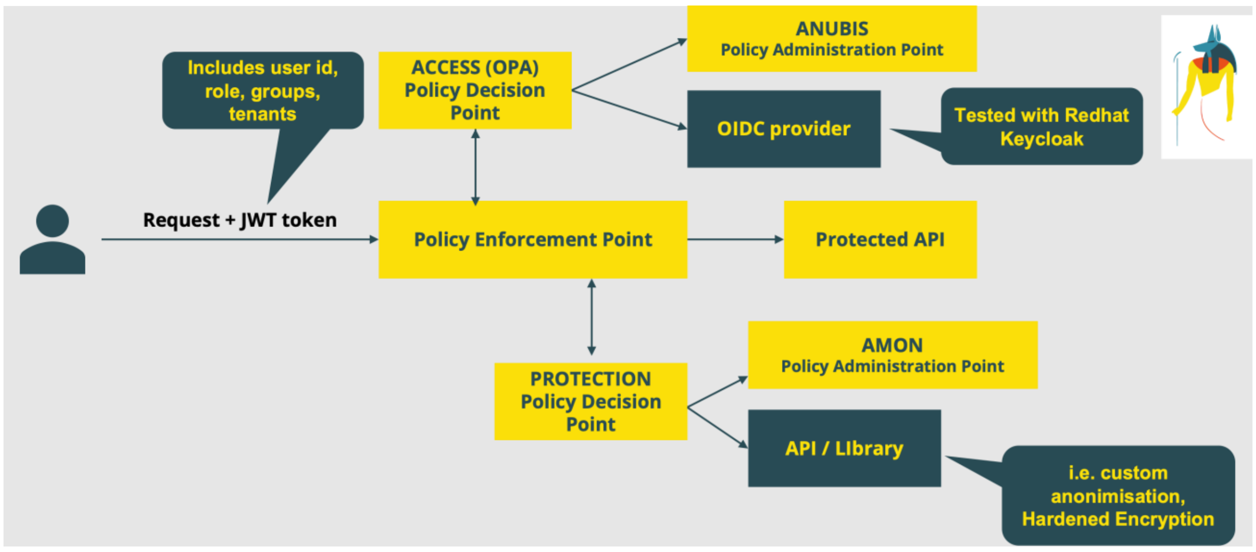

ANUBIS
Policy Administration Point
Includes user id, role, groups, tenants
ACCESS (OPA)
Policy Decision Point
OIDC provider
Tested with Redhat Keycloak
Request + JWT token
Policy Enforcement Point
Protected API
AMON
Policy Administration Point
PROTECTION
Policy Decision Point
API / Library
i.e. custom anonimisation, Hardened Encryption

Figure 6 - Logical view of the combined action of Anubis and Amon

A client requests for a resource via the PEP - implemented using an Envoy's proxy authz filter.

The PEP pass over the request to the PDP, while for the access to the data, the PDP is provided by Anubis, in Amon a dedicated engine provides enforcement of data protection policies (i.e. encrypt/anonymize attributes of data resources at rest or in transit);

The enforcement of data protection policies takes place according to the policies stored in the PAP, provided by the Data Protection Policy Management API;

Based on the access control and data protection policies, the client receives back the requested resource where data are encrypted/anonymised as by policy.

2.1.3.2 Interface description

Table 2 details the external interfaces:

Partner	Definition	Channel	Exchange

XLAB	Hardened Encryption	Direct Code Integration	Encryption libraries integrated inside Amon
IPN	Authentication	Direct Code Integration	JWT Token respecting the OIDC standard

Table 2 - External interfaces of the SADP component

2.1.3.3 Technical solution

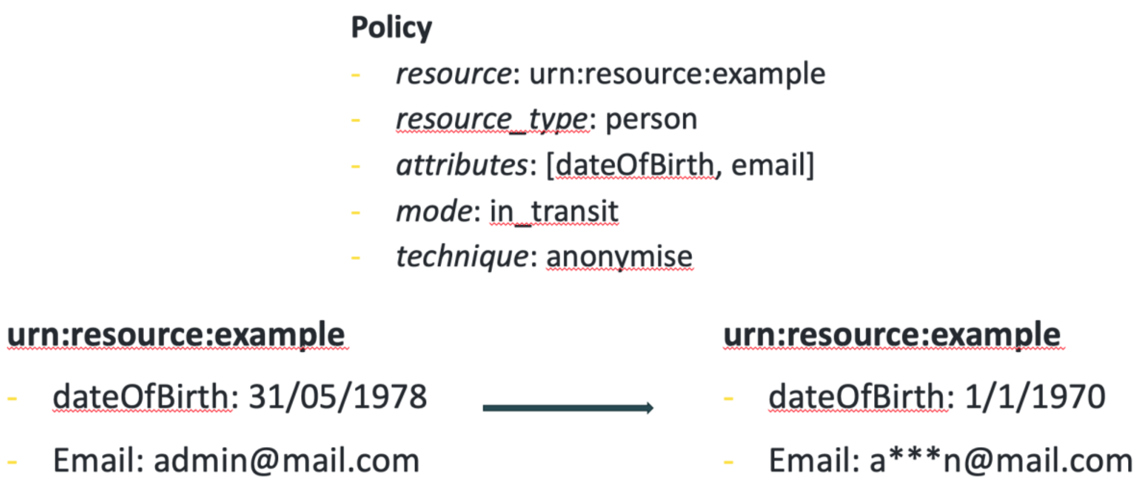
We present in the next sections the properties of the policies with an example and summarise how the API are structured.

2.1.3.3.1 *Policy Format*

The formal data protection policy specification is defined by an oc-acl vocabulary[11]. The internal representation is json-based and illustrated in the next section about the API.

- *resource*: The urn of the resource being targeted (e.g. urn:entity:x)
- *resource_type*: The type of the resource.
- *attributes:* The set of attributes the protection applies to
- *mode:* The protection mode applied (in transit, at rest)
- *technique:* The protection technique applied (e.g. anonymize, encrypt, …)

In Figure 7 we proved an example of the policy description and its action on a resource of example.



Policy

- *resource*: urn:resource:example
- *resource_type*: person
- *attributes*: [dateOfBirth, email]
- *mode*: in_transit
- *technique*: anonymise

urn:resource:example

- dateOfBirth: 31/05/1978
- Email: admin@mail.com

urn:resource:example

- dateOfBirth: 1/1/1970
- Email: a***n@mail.com

Figure 7 - SADP6 Policy example

2.1.3.3.2 *API specification*

We report in this section the API specification as available also in the online repository[12].

[11] https://github.com/orchestracities/anubis-vocabulary/blob/master/oc-acl.ttl
[12] https://github.com/orchestracities/amon/tree/master/amon-management-api

The Amon APIs depend on the FIWARE, in particular the policy management part is complemented by Tenant and Service management as supported by FIWARE APIs.

While policies are expressed in a generic way, and therefore it means that policies work also for other APIs beyond FIWARE, on the other hand FIWARE introduced a specific way to support multi-tenancy, and this API complies with FIWARE multi-tenancy model[13].

A tenant, also named as Fiware-Service, is further segmented into ServicePaths, i.e. hierarchical scoping of the resources part of a given tenant. For a more detailed discussion, you can read the related discussion in FIWARE Orion Context Broker documentation[14].

The API is composed by these main paths:

- **/v1/tenants** supporting definition of tenants (FIWARE Services) and paths (FIWARE Service Paths). Inherited from Anubis. NOTE: FIWARE Service Paths may be gone with NGSI-LD.
- **/v1/policies** supporting definition of data protection policies (linked to tenants and service paths). Under the hood, this path also creates and stores information linked to policies:
    - Encryption Modes (i.e. in transit or at rest)
    - Encryption Techniques (i.e. a given type of encryption or anonymization);

This is the current status of Amon:

- V0.1 available on GitHub: https://github.com/orchestracities/amon
- Key features completed:
    - Policy definition
    - OIDC with Keycloak

Dependences, libraries:

- W3C WAC
- W3C ODRL
- OAUTH2 and OIDC

Requirements:

- Python 3.9
- pipenv
- A database supported by sqlalchemy

License:

- APACHE 2.0.

Example:

- **Starts the API** (dev mode)

  ```
  $ pipenv install --dev
  $ pipenv shell
  ```

[13] https://fiware-orion.readthedocs.io/en/3.7.0/user/multitenancy.html
[14] https://fiware-orion.readthedocs.io/en/3.7.0/user/multitenancy.html

```
$ uvicorn amon.main:app --port 8075 --reload
```

note: custom logging for some reasons is not working:

```
uvicorn amon.main:app --port 8075 --reload --log-config ./logging.yaml --access-log
```

- **Test the API**

  ```
  $ pytest -rP
  ```

  Generate UML diagram for data models

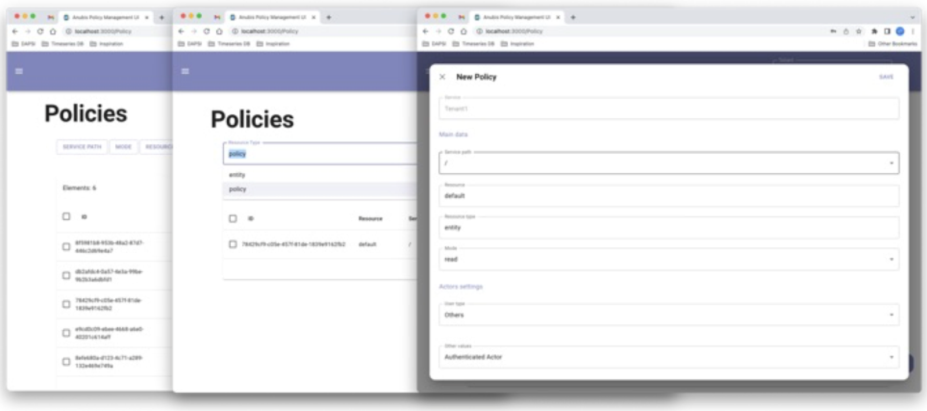
  ```
  $ python generate_uml.py
  ```

  API documentation

  Once the API is running, you can check it at: http://127.0.0.1:8075/docs

2.1.3.3.3 *Frontend design*

The front end for the specification of Amon's policies has been developed similarly to the one developed for Anubis and allows to all users, also non-technical ones, to easily specify polices to single attributes (Figure 8).



Service path: /
Resource: default
Resource type: entity
Mode: read
User type: Others
Other values: Authenticated Actor

Figure 8 - Example of GUI for per attribute policy and technique enforcement

2.1.4 The recommender

This section will go over the recommender service developed to work alongside Amon and provide recommendations for the policies to apply on the attributes for any given data entity.

2.1.4.1 The recommendation flow

At its core the recommender is a HTTP service component which is queried by the Amon frontend when a user views the policies applied to a given set of data attributes (Figure 9). For each attribute, a policy enforces a technique for encryption/anonymisation. The purpose of the recommender is to provide a recommendation for what are the "best" techniques to apply to that attribute in the given context.

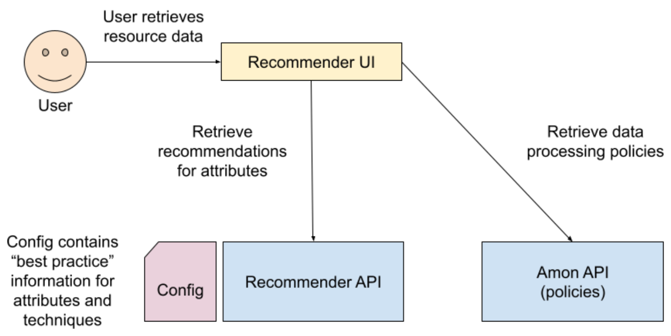
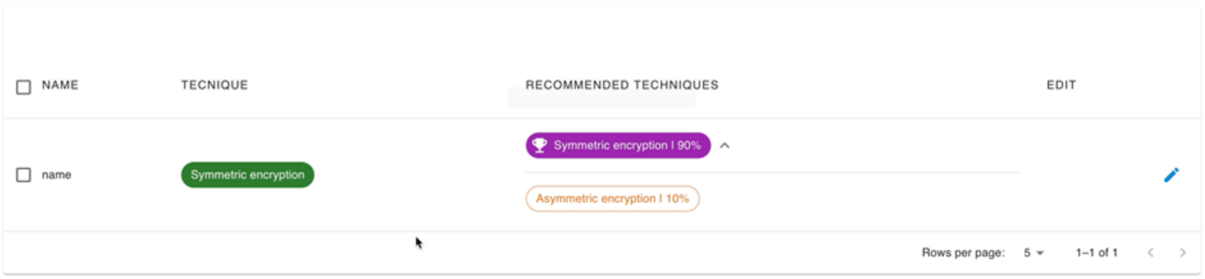

User retrieves resource data
User
Recommender UI
Retrieve recommendations for attributes
Retrieve data processing policies
Config contains "best practice" information for attributes and techniques
Config
Recommender API
Amon API (policies)

Figure 9 - Frontend and Recommender flow

A simple example: we have an attribute called "name", and the user wants to apply a policy to it. The recommender when queried about the attribute "name" and provided the context for that attribute (e.g. "medical patient"), will return a list of suggested techniques to enforce with a policy, like anonymisation or encryption (Figure 10).



NAME	TECNIQUE	RECOMMENDED TECHNIQUES	EDIT
name	Symmetric encryption	Symmetric encryption I 90% Asymmetric encryption I 10%	

Rows per page: 5 1–1 of 1

Figure 10 - Frontend view of a policy and the recommended techniques for it

The recommender will return multiple technique suggestions, each given a "score" to give the user an immediate idea of what the "best" options are.

The way the recommender provides these recommendations is based on a specific configuration file, which provides "guidelines" for which techniques to apply for a given set of attributes in specific domains. From these guidelines, the recommender can derive recommendations for a larger set of attributes, using mainly lexical semantical analysis (e.g. two different attributes with a similar meaning requiring the same technique).

The configuration is derived from general "best practice" techniques (encrypting sensitive medical data, anonymising names, …), currently manually curated using publicly available databases as a reference point (medical database models for instance). The option to automate the configuration and use a learning was investigated but could not be implemented within the framework of the project and could be explored into more details as future work in other projects.

2.1.5 Evaluation results

Amon is currently able to enforce encryption policies by leveraging the Hardened Encryption library to transform a given set of data attributes in transit. This includes being able to retrieve public and private keys to perform the task and applying the correct encryption policy on top of the enforcement policies. In addition, Amon can perform data anonymisation on a variety of data types (direct implementation), and supports 10+ policies for asymmetric and asymmetric encryption, anonymisation and pseudo-anonymisation, and functional encryption, leveraging the gofe libraries.

Regarding the recommender system, in addition to the implementation we have performed a baseline evaluation using our current configuration focusing primarily on medical domain attributes for patients (e.g. name, age, blood type, blood pressure, …). The evaluation is performed on a test set of data attributes, derived from real world medical database models, calculating the precision and recall of the recommender and the resulting F1 score (see KPIs).

Given the current configuration and a test data set of around 90 attributes, we can achieve an F1 score slightly above 0.8. This meets our previously established goal, but evaluation with larger data sets in multiple domains and further tweaking of the configuration would improve the accuracy of this test and future ones.

The user testing to assess the effectiveness of the recommender GUI was performed internally and allowed to reach a satisfactory level (80% of users showing positive evaluation), however lower than expected (90%). This is due to the need of more improvement cycles which could not performed within the framework of the project.

2.1.6 Future work

Future work outside the timeframe of WP3 might focus on further improving Anubis and Amon to work together, alongside further refinements of the recommender, including the previously mentioned approach towards learning models for its configuration and its GUI.

2.2 Federated AI

2.2.1 Overview

2.2.1.1 Description

Within the scope of the ARCADIAN-IoT project, Research Institutes of Sweden (RISE) has built dependable and privacy preserving Federated Learning (FL) capabilities are deployed in machine learning (ML)-based components (e.g., CTI and Behaviour Monitoring components). The proposed solution provides both source integrity (the guarantee that no malicious participant is involved in the process) and data integrity (privacy of raw data and local model updates is preserved). In addition, RISE has investigated and proposed solutions for the problem of having statistical and systematic heterogeneity of data in IoT environments, which usually determines an inaccurate and vulnerable training process.

Federated AI component has been developed as integrated module within those ARCADIAN-IoT components that provide ML models as their functionalities (e.g., CTI and Behaviour Monitoring component). It includes two subcomponents: data rebalancer and model resizing and sharing. Data rebalancer will provide a way to rebalance and fit non-Independent and Identically Distributed (non-IID) data to the framework; Model resizing and sharing subcomponent implements a communication-efficient and robust framework for model aggregation while preserving source integrity. This subcomponent accelerates and protects the local model from being attacked by adversarial attacks from malicious entities. Data integrity is provided by (i) the data rebalancer which only shares generated synthetic data and (ii) the model resizing and sharing subcomponent which uses standard FL paradigm to share only processed ML models.

2.2.1.2 Requirements

A recall of the high-level requirements that have been previously defined and provided in deliverable D2.4 [2] is given below.

Requirement 2.2.1 – Malicious behaviours from sharing entities to be detected: When an entity collaborating in the FL setup misbehaves during the learning process, it should be detected in order to invalidate the output.

Requirement 2.2.2 – Local ML model to be lightweight: The models that are generated locally by the CTI should be lightweight, as this will make the FL more efficient in the federated model computation.

Requirement 2.2.3 – At least three heterogeneous devices/entities: The Federated AI mechanisms should support the training among at least three heterogeneous devices and entities.

2.2.1.3 Objectives and KPIs

The main objective is to provide a dependable privacy preserving classifier based on FL which (i) incorporates classical data balancing techniques traditionally used for dealing with imbalanced date in centralized ML, and (ii) ensures source and data integrity in the learning phase. With the aim of fulfilling the project's objectives, the Federated AI module has been evaluated against the following KPIs:

KPI scope	
Provide a new data rebalancing techniques for federated learning setting outperforming state-of-the-art methods (K-SMOTE, SMOTE, up/down-sampling) in term of accuracy and efficiency	
Measurable Indicator	
False Positive rates (FPR), False Negative rates (FNR), Precision, Recall, number of epochs per rebalancing cycle	
Benchmarking	
K-SMOTE [12], SMOTE [3,4], Upsampling [1], Downsampling [2]	
Target value	***Achieved value***
FPR < 5%, FNR <5%, Precision >99%, Recall >95%, #epochs per rebalancing cycle: < 50 epochs	FPR ≃1%, FNR ≃2%, Precision ≃99%, Recall ≃97%, #epochs per rebalancing cycle ≃ 30 epochs

KPI scope	
Improve communication efficiency in federated learning	
Measurable Indicator	
AUC and F1-score, and number of epochs required to converge	
Benchmarking	
Random-K Sparsification [6], Top-K Sparsification [5]	
Target value	***Achieved value***
AUC > 99%, F1-score > 94% #epochs to converge < 35 epochs	AUC ≃ 100%, F1-score ≃ 96% #epochs to converge < 15 epochs

KPI scope	
Improve robustness in federated learning against model and data poisoning attacks	
Measurable Indicator	
F1-score and False Negative Rate (computed with data and model poisoning attack)	
Benchmarking	
Federated Average (baseline) [9], Median, Trimmed-Mean [10], Krum [11]	
Target value	***Achieved value***
F1-score (under data poisoning attack) > 93% FNR (under data poisoning attack) < 12% F1-score (under model poisoning attack) > 85% FNR (under model poisoning attack) < 20%	F1-score (under data poisoning attack) = 97.63% FNR (under data poisoning attack) = 4.27% F1-score (under model poisoning attack) = 93.94% FNR (under model poisoning attack) = 9.71%

2.2.2 Technology research

2.2.2.1 Background

First, statistical heterogeneity can arise from non-IID data collection in IoT networks, because of the FL setups, the number of the data points or data distribution may vary significantly across devices or clients, which would degrade the performance of the model. Imbalanced data is a common issue in real-world classification tasks. It refers to the problem that one class is heavily under-represented compared to the other class, in a two-class classification problem. It affects the performance of the classifier since prediction leans towards the majority class, and the minor class is usually wrongly classified. This situation is typical for many cyber security applications, including anomaly, attack, or fraud detection, where datasets are often quite imbalanced. The state-of-the-art data rebalancing techniques include: (i) oversampling [1], (ii) under-sampling [2], and (iii) Synthetic Minority Oversampling Technique (SMOTE) [3,4]. Oversampling and under-sampling are two classic techniques to address this issue. Nonetheless, the application of these techniques has not been explored in contexts for FL.

Second, complete training round of FL includes uploading and downloading between every client and the central aggregator. Expensive communication cost is a bottleneck for many FL deployments, especially when using large DNNs that contains millions of parameters, that are shared during the training process between the clients and the aggregator. Therefore, communication overhead is one critical challenge that needs to be addressed. There are several state-of-the-art solutions including: (i) model sparsification [5,6], (ii) quantization [7], and (iii) parallelization [8]. Among these three methods, we found model sparsification is the most promising techniques. The main idea of model sparsification is to only communicate a reduced number of components of the model's gradients (or parameters) in order to improve the efficiency, but, at the same time, achieving a good estimate of the global gradient so that the performance is not significantly affected. Two state-of-the-art sparsification methods: (i) Top-K sparsification [5] and (ii) random-K sparsification [6]. Top-K sparsification is to choose top-k parameters with the highest absolute value from the gradients and assign the rest of the components as the residuals; random-K, literately, is to choose random k numbers of parameters. These two methods seem promising, they still hold some shortcoming such as slow convergence.

Last but not the least, standard aggregation algorithms, such as Federated Average [9], are extremely vulnerable to data and model poisoning attacks. Thus, a single adversary can entirely compromise the training process. Robust aggregation techniques for FL have become a relevant and popular research topic for FL, given the impact that these attacks can have in practical deployments. Data poisoning attack refers to the attacker manipulates the training dataset with aim to mis-train the classifier; model poisoning attack refers to the attacker manipulates the model parameters, or the updates shared with the central aggregator. Several state-of-the-art defence techniques have been proposed: (i) *Median* and *Trimmed-Mean* proposed in [10] are two aggregation algorithms relying on robust statistics, and (ii) *Krum* [11] considers the similarity of the gradients of model updates from all the clients in every iteration.

2.2.2.2 Research findings and achievements

During the course of WP3, RISE has conducted research and development activities on two main subcomponents: (i) the data rebalancer, which aims at rebalancing non-IID and imbalanced data in a federated set-up, and (ii) the model resizing and sharing subcomponent. The research activity can be summarized as follow:

- Studied the issues related to the statistical heterogeneity (non-IID data) within Federated AI and IoT network, which usually causes a degradation of training the models.
- Explored and analysed three state-of-the-art data rebalancing methods.
- Designed and implemented a new potential solution for data rebalancing.
- Evaluated the proposed data rebalancing approach with two open source IoT datasets.
- Explored and analysed state-of-the-art communication-efficient aggregation rules.
- Explored and analysed state-of-the-art robust aggregation rules.
- Designed, implemented and evaluated a novel communication-efficient and robust aggregation rule for FL.

2.2.2.2.1 Subcomponent: Data Rebalancer

After understanding the challenges and situations of previous works, RISE has studied and developed an adaptive data rebalancing technique, which can be used in peer-to-peer FL, and for non-IID data. Starting from the K-SMOTE [12], a variation of the original SMOTE for IoT settings, RISE has defined a new technique able to generate more complex synthetic points to share some of them with other participating clients. Figure 11 depicts how the proposed Data rebalancer subcomponent works.

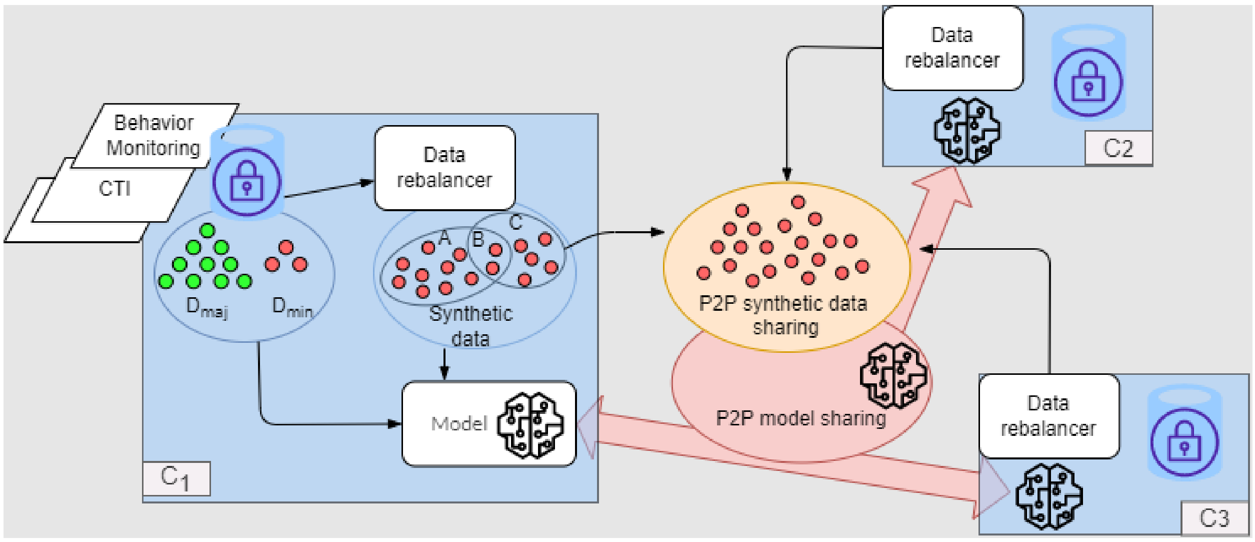


Figure 11 - Overview of Data rebalancing approach

Within an IoT network, the Data rebalancer can be deployed on the edge devices, referred to as clients in FL setups. The edge device can be, for example, a gateway that includes Behaviour Monitoring capabilities in order to detect any malicious activities. In our proposed solution, there is no central node required to participate to or orchestrate the training process. Overall, the entire process that involves the data balancer consists of three phases: (i) synthetic data generation, (ii) model optimization, and (iii) data and model sharing. In the first phase, the training data available at a client are rebalanced by an over-sampling technique (i.e., augmenting the number of data points from the minority class) which generates synthetic data points from the genuine data points in the minority class. The synthetic data points are randomly split into three subsets (A, B, and C). The synthetic data points in the set $(A \cup B)$[15] are merged with the genuine data points so as to obtain a balanced dataset which is fed into the local ML model stored by the client. Then, the ML model is trained with the merged balanced dataset. Finally, the resulting model and synthetic data points in the set $(B \cup C)$ are shared with a subset of the connected clients. It is important to note that the clients are not sharing directly training data points, but a mix of synthetic data points which are partially not used for training the local model.

Again, the cause of non-IID phenomenon is heterogeneity of various IoT devices, and this phenomenon degrades the model's performance in FL. It is bound to sacrifice some privacy to solve the problem of non-IID. Instead of having an auxiliary dataset maintained by the central node, the algorithm shares only some artificial data between some participants to help their local data rebalancing. The challenge is how to preserve the privacy of data when sharing the artificial data which is generated based on the raw data. SMOTE, as a state-of-the-art rebalancing method, has been used in many imbalanced data problems. It generates synthetic data by linear interpolation of samples in the minority class. However, it has the risk that the genuine data points are easy to be inferred when synthetic data is shared with other peers. To reduce the risk of privacy breaches, RISE defined and developed HSphere-SMOTE, an enhanced version of the SMOTE, suitable for FL sessions in IoT scenarios.

```
Algorithm 1: Algorithm of HSphere-SMOTE
  Input :
      $D_{min}$: set of the data points in the minority class
      $d_{min}$: number of data points in $D_{min}$
      $d_{sam}(< d_{min})$: number of data points to use for sampling
      $d_{syn}$: number of synthetic data points to generate
  Output:
      $S$: Set of synthetic points
1  $R \leftarrow$ Set of $d_{sam}$ randomly picked samples in the minority class
2  $d_{pe} = d_{syn}/d_{sam}$
3  $S = []$
4  for $x_i \in R$ do
5      $N(\subseteq D_{min}) \leftarrow$ Set of nearest neighbors for $x_i$
6      $S_{x_i} = []$
7      $x_{c_i} = x_i + \frac{1}{|N|} \sum_{x_j \in N} (x_j - x_i) \times rand.beta(2,2)$
8      $r = \max_{x_j \in N} \{(x_j - x_i)\}$
9      $S_{x_i} \leftarrow$ Sample $d_{pe}$ points from pdf $P_{HSphere}\{x, x_{c_i}, r\}$
10     adjoin $S_{x_i}$ to $S$
11 end
  ▷ $P_{HSphere}\{x, a, r\}$ : probability density function over the volume of the
  hyper-sphere centered at $a$ with radius $r$.
  ▷ $rand.beta(\alpha, \beta)$: random value drawn from the Beta distribution with two
  positive shape parameters $\alpha$ and $\beta$.
```

Algorithm 1

Algorithm 1 describes the whole process of HSphere-SMOTE in detail. First, the input to the algorithm consists of the dataset that includes labelled data from the minority class, and several hyper-parameters including the number of base data to re-sample and the number of synthetic points to be created. The output is the final set of synthetic points. HSphere-SMOTE first randomly

[15] The symbol ∪ is employed to denote the union of two sets.

picks d_{sam} base data points (line 1). For each base data point x_i, the algorithm samples d_{pe} synthetic points from a probability density function $P_{HSphere}$ with uniform distribution over the volume of a hypersphere centered at x_{c_i} and with radius equals to the maximum distance between x_{c_i} and any neighbours of x_i.

2.2.2.2.2 *Subcomponent: Model Resizing and Sharing*

RISE has also worked on the design and development of a communication-efficient and robust aggregation rule which can be used in peer-to-peer FL and for non-IID data. After surveying the state-of-the-art techniques for improving communication efficiency and robustness, RISE has designed a novel aggregation rule, called SparSFA, that ensures the smoothness the training process of FL. It considers both the communication efficiency and utility, and it is robust enough to defend against data or model poisoning attacks. Figure 12 depicts how the proposed model resizing and sharing subcomponent works.

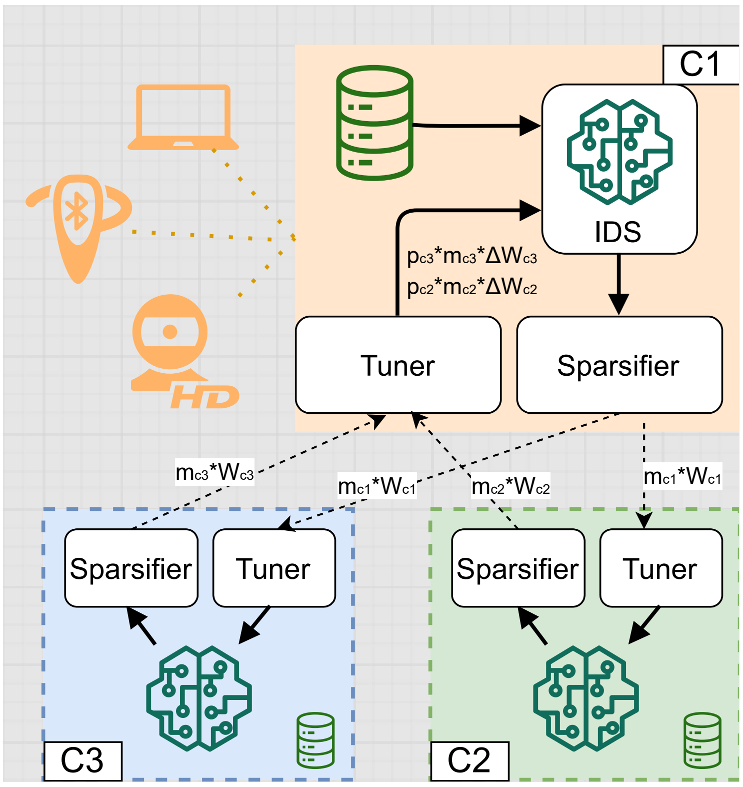


Figure 12 - The overview of subcomponent (Model resizing and sharing)

Same design as the data rebalancer, SparSFA can be employed by ML-based IDS on edge devices which are regarded as the clients in FL setup. As shown in Figure 12, overall, the whole subcomponent can be divided into three parts: (i) model sparsification, (ii) clients' weight tuning, and (iii) optimization of the local model for intrusion detection. First, the local model is initially trained at each client with its local dataset. When the other neighbouring clients call for an update, Sparsifier module zeros out a significant number of the parameters, compressing the information to be shared by focusing on the most relevant parameters. This not only reduces the communication overhead, but also conceals some less-essential information of the model that could also lead to privacy leaks. Next, before updating the local model, the tuner module adjusts the score of model updates received from the connected neighbours of the client. Finally, the local model is updated by aggregating the tuned parameters of the connected neighbours of the client, and starts the next round of local training. This collaborative learning task runs iteratively until some level of convergence is attained or a maximum number of training rounds is reached.

After designing the subcomponent, we first focus on developing Sparsifier. We decided to utilize model sparsification method and further improve Top-K sparsification proposed by Aji et al. [5]. The idea of model sparsification is to share small subset of the model parameters, and zero out the rest. Top-K sparsification zeros out k parameters with the largest magnitude in the gradient are selected. However, the shortcoming of Top-K sparsification is that it is possible some specific parameters, which often contribute to large magnitudes in the gradient can dominate the sparsification process and other relevant parameters may not be shared with the peers. In addition, only sharing a specific part of model will limit the performance of the local model on the different clients, especially in non-IID scenarios. To overcome this limitation, we propose to improve TopK sparsification by means of adding momentum to the residual to restrict the impact of the current gradient, stabilizing the training, and improving the performance. We name the algorithm MomTopK.

```
Algorithm 2: Algorithm of MomTopK
  Input  : The parameters of the local model in $t^{th}$ iteration for the $i^{th}$
           client: $\mathcal{W}_{t,i}$, Hyper-parameters: the forgetting factor $\beta$, and
           the sparse ratio $r$
  Output: Sparse parameters of the model: $\hat{w}_{t,i}$
1 Initialize mask and residual to be zero: $m_0 = \mathcal{R}_0 = 0$
2 $\nabla F(\mathcal{W}_{t,i}) \leftarrow Gradient(\mathcal{W}_{t,i})$
3 $\mathcal{R}_{t,i} = \beta\mathcal{R}_{t-1,i} + (1-\beta)\nabla F(\mathcal{W}_{t,i})$
4 $k \leftarrow r * numel(\mathcal{R}_{t,i})$
  /* The number of parameters to be zeroed out                    */
5 Generate top-k mask $m_t$ from $\mathcal{R}_{t,i}$
6 $\mathcal{R}_{t+1,i} = \mathcal{R}_{t,i} - m_t \circ \nabla F(\mathcal{W}_{t,i})$
7 $\hat{w}_{t,i} = m_t \circ \mathcal{W}_{t,i}$
```

Algorithm 2

Algorithm 2 described the detailed process of MomTopK. The main input is the parameters of the local model that will be shared with the neighbors and the two hyperparameters, the forgetting factor and sparse ratio. We first initialize the mask and the residual set. At t communication round, the residual is a trade-off summation of the residual from the previous iteration with accumulated gradients and the gradient obtained from the current gradient. (line 3). As shown in lines 4 and 5, the number of parameters to share is obtained by the sparse ratio, and the mask, which is a binary tensor, is generated based on the k largest values of the residual. Finally, the sparsified model's parameters are obtained as shown in line 7. The residual for the next round is obtained by the subtracting the masked gradient from current residual as described in line 6 of the algorithm.

In order to defend against adversarial attacks, we design a robust federated aggregation rule by assigning trust scores to the client. It controls the contributions of its shared model. The trust score is computed according to their trustworthiness in the model updates provided at each training round relying on a set of five metrics. It includes data size, data variance, connectivity, model similarity, and model divergence. We assume the larger dataset one peer holds, the more relevant information it can provide. We also assume higher variance will correspond with a more diverse dataset. If the peer connects to many other clients, it means this specific peer is in a good position in the network topology. In one iteration, we measure the model similarity and model divergence between the original model, and the sparsifed model contributed from the peer. The more similar between two models, the safer the peer is assumed in the current iteration. Contrary to model similarity, when two models are divergent from each other, then this divergence brings more unpredictability to the model that is going to be updated.

When client c_d calls for update in t^{th} iteration, it assigns a score to each of its connected clients in the set $N_{c_d} = \{s: c_s$ connected to $c_d\}$ based on these five metrics. The score to the client $s \in N_{c_d}$ is denoted by $p^{s,d}$ and is defined as a linear combination of the five metrics where the last metric i.e., divergence, is regarded as a penalty to the linear combination. Thus, the score $p^{s,d}$ in t^{th} iteration can be obtained as:

$$p_t^{s,d} = a_1 \cdot \text{DataSize}\,(c_s) + a_2 \cdot \text{DataVar}\,(c_s) + a_3 \cdot \text{Conn}\,(c_s)$$
$$+a_4 \cdot \text{Cos}_t\left(\widehat{w}_{t,s}, \mathcal{W}_{t,d}\right) - a_5 \cdot \text{Div}_t\left(\widehat{w}_{t,s}, \mathcal{W}_{t,d}\right)$$

where $a_1 \cdots a_5 \in \mathcal{A} \in \mathbb{R}^5$ are the scalar coefficients, and a_5 is regarded as the penalty term. In order to find the optimal set of $\mathcal{A}$ to correctly score the neighbor clients in set N_{c_d}, we use Bayesian optimization to tune $p_t^{s,d}$. This black-box optimization technique relies on Bayes Theorem to direct the search in order to find the minimum or maximum of an objective function [23]. In our case, we use the loss on the local training dataset of the client c_d to learn the best coefficient combination. We build a probabilistic model of the objective function as the surrogate function on every client. We call this as the tuner module, as depicted in Figure 12, based on Bayes Theorem.

2.2.2.3 Produced resources

The developed code for Federated AI component, including two sub-components, written in Python is available on the GitLab repository https://gitlab.com/arcadian_iot/federated-ai.

Algorithm Process:

(1) Initialize every local model and weight the client by the data size and data variance.
(2) Optional: Rebalancing the data: During each training epoch, the synthetic data are generated by calling HSphereSMOTE. The data are ready to be used for the local training and shared among peers.
(3) Local training: The local model on each client is trained with its own data combined with synthetic data (its own and gathered from the peers)
(4) Sparsification: The local model is sparsified by the proposed Mom-topK, and ready to be shared.
(5) Share the local models with peers: Weighted Federated Averaging
(6) The received model is tuned according to 5 metrics: peer's data_size, data_variance, connectivity, model similarity, model divergence.
(7) Update the local models.

Structure of the repository:

- 0_data:
  - IoT:
    - 0_down.sh: the shell script to download the required dataset.
    - data_process.py: the script that splits the dataset and save the data to the pickle files.
      - Two arguments:
        - –-fiveclient, Boolean. It splits the dataset to 5 subsets that is designed for example experiments.
        - –num_client, integer, optional. It splits the dataset to any number of subsets. (It cannot be specified with --fiveclient)
  - UNSW:
    - 0_down.sh: the shell script to download the required dataset.
    - data_process.py: the script that splits the dataset and save the data to the pickle files.
      - Two arguments:
        - –-fiveclient: Boolean: It splits the dataset to 5 subsets that is designed for example experiments
        - –num_client: integer, optional: It splits the dataset to any number of subsets. (It cannot be specified with –fiveclient)

- Core:
  - HSphereSMOTE.py: the main file that includes the main functions of the data rebalancer.

  - P2P_Aggregation.py: the main file that includes functions of proposed methods for model sparsification and defenses.
  - SOTA_attacks.py: the file that contains implemented state-of-the-art attacks, including label flipping attack, data noise attack, objective function poisoning attack, and Byzantine attack. The attack can be selected to proceed in the main python script with assigned argument.
  - SOTA_Defense.py: the file that contains implemented state-of-the-art defenses, including Krum, Trimmed-Mean, Median
- Utils:
  - Data_distribute.py: the file that includes the functions for distributing data points and normalizing them.
  - Models.py: the file that contains the model structure.
  - Train.py: the main file that contains the functions for model training.
  - Utils.py: the file that contains regular functions.
- Main.py: The main python script to run the example program with given dataset and proposed aggregation framework with specific arguments.

2.2.3 Design specification

2.2.3.1 Logical architecture view

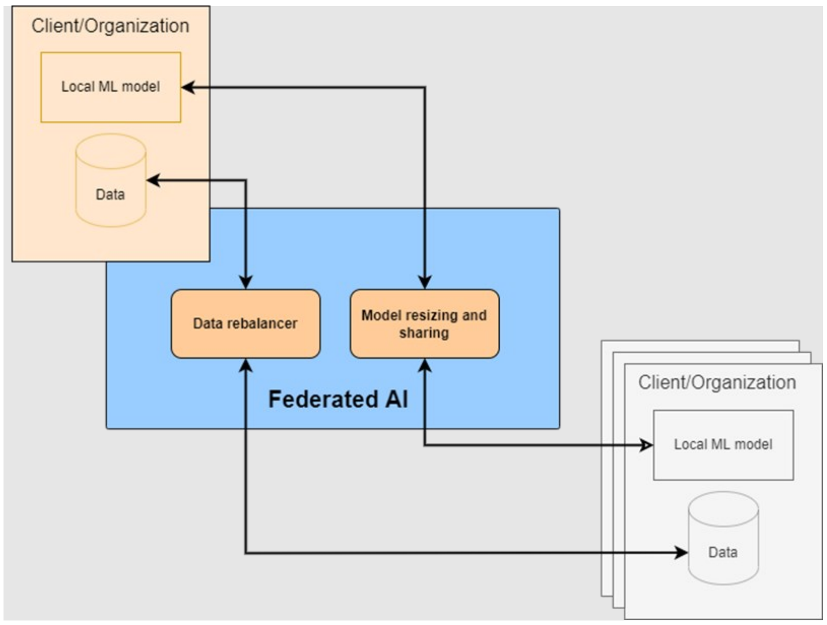


Figure 13 - Component architecture of Federated AI

The component further provides accurate and trustworthy framework by addressing challenges of federated learning, such as imbalanced and non-IID data, and adversarial attacks. Federated AI includes two sub-components: data rebalancer and model resizing and sharing as shown in Figure 13. This component directly interacts with Cyber Threat Intelligence (CTI) component.

- **Data rebalancer**: provides a way to rebalance and fit non-IID data to the framework. It directly interfaces with the local database within the client component.
- **Model resizing and sharing:** provides a communication-efficient and robust framework for model aggregation. It directly interfaces with the local ML algorithm within the client component.

2.2.3.2 Sequence diagrams

Figure 14 describes the sequence diagram of the Federated AI component which includes the functions for data rebalancing, sparsification, tuning, and local training. As a first step, the central server transfers the model configuration to all clients in the federated setup. Starting from the local dataset, the data rebalancer generates a set of synthetic points to rebalance the distribution between classes. The synthetic points are exchanged between neighbour clients only. The new dataset made of local original and synthetic data, and neighbour synthetic data are used to train the local model and to run the sparsification algorithm. The generated sparsified models are exchanged between neighbour clients. Finally, the local model is updated by aggregating the tuned parameters of the connected neighbours and starts the next round of local training. All these steps are part of a loop which is iteratively executed until convergence is reached.

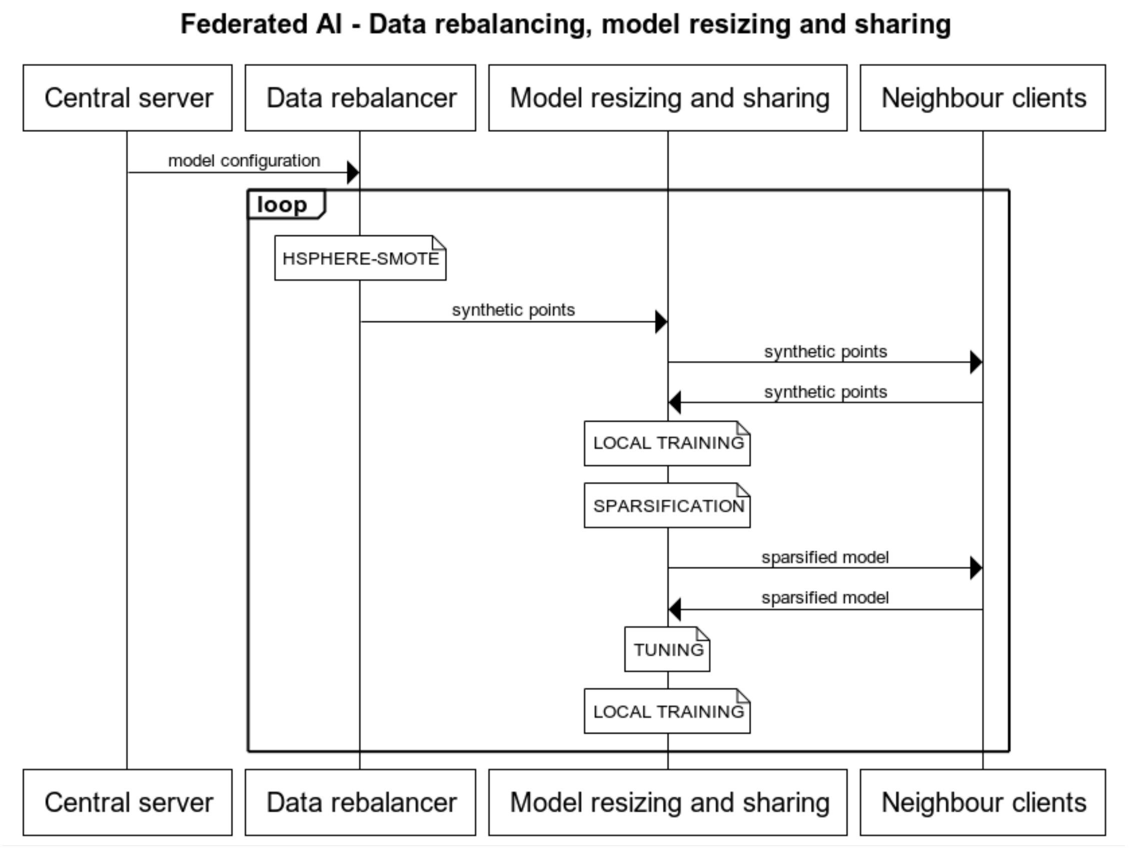


Figure 14 - Sequence diagram

2.2.3.3 Interface description

The external interfaces are the following:

- Behaviour Monitoring: Access to Federated AI libraries for data rebalancing; the component will be used to balance the two classes (normal and malicious behaviour) for IoT device intrusion detection.
- Behaviour Monitoring: Access to Federated AI libraries for Communication-efficient and robust aggregation rule; the component will be used to reduce the communication overhead in the federated communication and in parallel make the models robust against data and model poisoning attack.
- CTI: Access to Federated AI libraries for Communication-efficient and robust aggregation rule; the component will be used to reduce the communication overhead in the federated communication and in parallel make the models robust against data and model poisoning attack.

Table 3 below details the external interfaces:

Partner	Definition	Channel	Exchange
IPN	Behaviour Monitoring	Direct Code Integration	Data Rebalancing Libraries
IPN	Behaviour Monitoring	Direct Code Integration	Communication-efficient and robust aggregation rule Libraries
RISE	CTI	Direct Code Integration	Communication-efficient and robust aggregation rule Libraries

Table 3 - External interfaces of the Federated AI component

2.2.3.4 Technical solution

2.2.3.4.1 API specification

Federated AI functionalities are provided as a set of libraries that are directed integrated as part of the ML-based components (behaviour monitoring and CTI component).

2.2.4 Evaluation and results

We have evaluated HSphere-SMOTE for data rebalancing in federated set-ups with two open-source datasets. One is N-BaIoT [13], which consists of real-world network traffic flows from 9 commercial IoT devices. This dataset is collected for detecting Botnet in IoT network. It captures both benign traffic and malicious traffic carried by 2 botnets (Mirai, BASHLITE). We labelled the malicious traffic from Mirai as abnormal traffic. The other dataset is UNSW BoT-IoT [14], which includes data collected in Cyber Range Lab in UNSW. They simulate the network behaviour of devices in IoT network that is under Botnet attack. We measured False Negative Rate (FNR), False Positive Rate (FPR), Precision at recall 75%, and Recall at precision 75% on HSphere-SMOTE and compare its performance with every mentioned baseline's in two scenarios: (i) IID and imbalanced data, and (ii) non-IID and imbalanced data. The empirical results show HSphere-SMOTE outperforms the other baselines.

RISE has also evaluated SparSFA performance on communication efficiency and its robustness against poisoning attacks. We first measure communication efficiency, that is MomTopK of SparSFA, with the assumption that all clients are benign and no attack takes place. We compare MomTopK with other two state-of-the-arts methods: Random-K [6] and Top-K[5]. The results show that MomTopK can converge within 15 epochs with almost 100% of AUC and 96% of F1-score. It outperforms than the other two baselines.

We second measure its robustness in 5 different scenarios, (i) 5 clients with 1 adversary in complete graph, (ii) 5 clients with 1 adversary in incomplete graph, (iii) 5 clients with 1 adversary in a chain, (iv) 10 clients with 1 adversary in random network topology, and (v) 20 clients with 3 adversaries in random network topology. Our experiments also show that SparSFA is robust to 4 different data and model poisoning attacks, even outperforming other robust aggregation schemes like Krum [11], Trimmed-Mean and Median [10]. In all attack scenarios, SparSFA shows a F1-score above 95% and a FNR below 5%, even in scenarios with data imbalance, making it suitable for distributed IDS in IoT environments.

2.2.5 Future work

Future work outside the timeframe of WP3 could potentially concentrate on advancing the secure communication of Federated AI. This could involve incorporating and utilizing techniques like homomorphic encryption and multi-party computation. Additionally, efforts may be directed towards developing robust aggregation methods that can protect against privacy attacks. For instance, preventing the inference of the private training set when a malicious attacker queries the trained FL model.

3 SECURITY PLANE

3.1 Network Flow Monitoring

3.1.1 Overview

3.1.1.1 Description

Within the scope of the ARCADIAN-IoT project, the University of West of Scotland (UWS) has developed the Network Flow Monitoring (NFM) component that acts as an enhancement of existing Network Intrusion Detection Systems (NIDS), such as Snort,6 to achieve the detection of known malicious Distributed Denial of Service (DDoS) along the entire infrastructure of the 5G network. This has been achieved by the contribution of two different subcomponents: (I) a NIDS with an updated set of rules for known DDoS attacks, and (II) the Security Flow Monitoring Agent (SFMA) that acts as a wrapper for the NIDS alert and provides fine data information about the malicious flow detected. This component provides support not only for traditional IP networks, but also for the overlay networks currently used in cloud infrastructures employing overlay/encapsulation protocols such as Virtual Extensible LAN (VxLAN), Generic Routing Encapsulation (GRE), Generic Network Virtualization Encapsulation (GENEVE), those currently used in enterprise infrastructures such as VLAN, and those currently used in cellular and IoT mobile operator networks such as GTP used in LTE-M and NB-IoT.

3.1.1.2 Requirements

A recall of the high-level requirement that has been previously defined and provided in deliverable D2.4 [2] is given below.

Requirement 3.1.1 – IoT Network Detection: The flow monitoring component will have the capabilities to perform the detection of DDoS attacks in any network segment of the IoT infrastructure, triggering the associated alert.

3.1.1.3 Objectives and KPIs

KPI scope	
Provide new alert metadata information about IoT and 5G flow structure, within the supported network segments with a number of >= 4 (Edge, Core, RAN and Transport).	
Measurable Indicator	
Number of network segments supported to get the metadata information.	
Baseline	
Supporting only Core network segment (value: 1)	
Target value	***Achieved value***
>= 4 (Edge, Core, RAN and Transport)	= 4 (Edge, Core, RAN and Transport)

KPI scope
Provide transversal detection capabilities to protect, simultaneously tenant infrastructure and the infrastructure provider by supporting >=4 encapsulations and tunnelling protocols widely used in overlay networks inherent to 4G/5G IoT mobile infrastructures such as VXLAN, GRE, GENEVE and GTP.
Measurable Indicator
The number of supported encapsulations and tunnelling protocols widely used in overlay networks inherent to 4G/5G IoT mobile infrastructures.
Baseline

Supported 0 encapsulation and tunnelling protocols	
Target value	***Achieved value***
>= 4 (VXLAN, GRE, GENEVE, GTP)	= 4 (VXLAN, GRE, GENEVE, GTP)

3.1.2 Technology research

3.1.2.1 Background

The 5GS (5G System) architecture presents several different stakeholders that are involved in the provisioning of 5G network resources, as presented in the View on 5G Architecture by the 5G PPP (5G Public Private Partnership). A key role in the provision of 5G services is that of the Service Provider (SP), which interacts directly with service customers and obtains and orchestrates resources from Network Operators, VISPs (Virtualisation Infrastructure Service Providers) and DCSPs (Data Centre Service Providers), collectively referred to as infrastructure providers. Therefore, each of these stakeholders will have a different purpose in terms of acting on the data flow that is happening in their infrastructure (either physical or virtualised). This means that, along the entire route that a data flow must take, from its creation in the IoT device to its destination, passing through the entire 5G communications infrastructure, it will have variations in its morphology, despite its invariability in content. To achieve this, various encapsulation mechanisms are used to isolate traffic from each of the tenants that may be installed on the same infrastructure. Virtual Extensive LAN (VxLAN) is one of the examples, which is used to create the so-called multi-tenancy, isolating the traffic for each of the tenants which are occupying the infrastructure. This technology allows the creation of a first encapsulation of the data flow to achieve such isolation which will be used to identify the tenant. GTP is used as a second encapsulation, according to the standards of the 5G, allowing end-user mobility (Figure 15).

The research carried out by UWS until this point within ARCADIAN-IoT project has revealed that the closest state of the art associated to the Network Flow Monitoring component is the one related to Network Intrusion Detection System (NIDS). These systems usually perform autonomous inspection of traffic and notifies via an alert, that the network is being compromised by a cyber-attack. The main problems with these tools are:

- They usually work with traditional IP traffic. Hence, leaving behind the multiple overlay network encapsulations expected in 4G/5G-IoT systems (e.g. VXLAN, GRE, GTP, etc.).
- They do not provide an extended North Bound Interface with a standardised reporting system to send the fine-grained metadata information from the inner and overlay headers regarding the malicious flow.

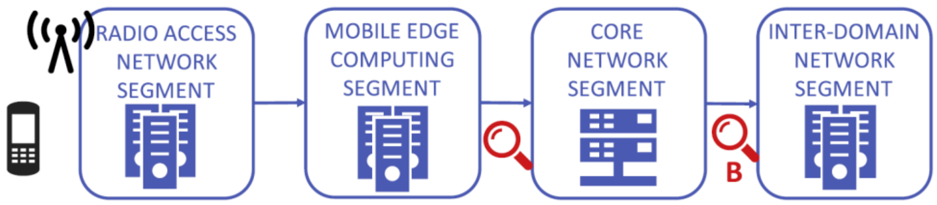
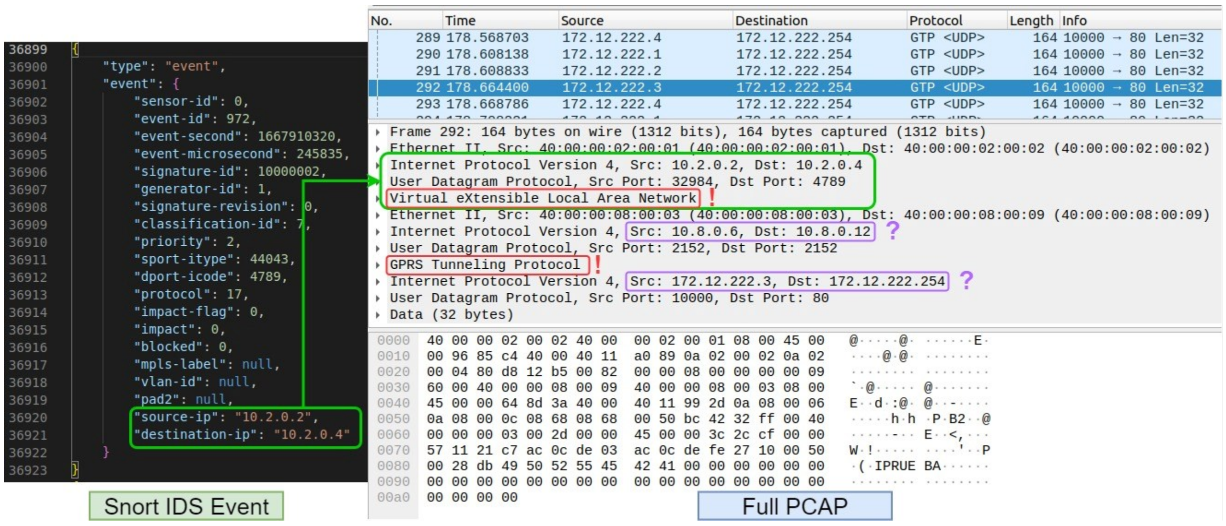

RADIO ACCESS NETWORK SEGMENT
MOBILE EDGE COMPUTING SEGMENT
CORE NETWORK SEGMENT
INTER-DOMAIN NETWORK SEGMENT
B

Figure 15 - 5G multi-stakeholder network segments

3.1.2.2 Research findings and achievements

Traditional signature-based NIDS such as Snort lack of 5G infrastructure and network information. This depicts a major missing point for all network infrastructure providers, that are directly affected in the moment their physical and virtualized infrastructures are threatened and need to be protected. All overlay network information stated in the paragraph above is important when the detection of a threat takes place. Fine grain metadata should be provided at the flow level, so the consequent actors of the protection capabilities are able to determine the idoneal place where to take the mitigation. Figure 16 shows an example of the information that a Snort IDS Event is able to retrieve (see left in Figure 16). As shown in the figure, the NIDS only gets the last overlay encapsulation, related in this case to the Virtual eXtensible Local Area Network (VXLAN) (see right in Figure 16). In the figure it can also be noticed that a nested encapsulation of GPRS Tunnelling Protocol (GTP) is also used by the 5G system, with more information related to the network flow. All this information is crucial when the system needs to find the best action in order to mitigate an attack.



```
{
    "type": "event",
    "event": {
        "sensor-id": 0,
        "event-id": 972,
        "event-second": 1667910320,
        "event-microsecond": 245835,
        "signature-id": 10000002,
        "generator-id": 1,
        "signature-revision": 0,
        "classification-id": 7,
        "priority": 2,
        "sport-itype": 44043,
        "dport-icode": 4789,
        "protocol": 17,
        "impact-flag": 0,
        "impact": 0,
        "blocked": 0,
        "mpls-label": null,
        "vlan-id": null,
        "pad2": null,
        "source-ip": "10.2.0.2",
        "destination-ip": "10.2.0.4"
    }
}
```

```
No.   Time        Source         Destination      Protocol    Length Info
  289 178.568703  172.12.222.4   172.12.222.254   GTP <UDP>      164 10000 → 80 Len=32
  290 178.608138  172.12.222.1   172.12.222.254   GTP <UDP>      164 10000 → 80 Len=32
  291 178.608833  172.12.222.2   172.12.222.254   GTP <UDP>      164 10000 → 80 Len=32
  292 178.664400  172.12.222.3   172.12.222.254   GTP <UDP>      164 10000 → 80 Len=32
  293 178.668786  172.12.222.4   172.12.222.254   GTP <UDP>      164 10000 → 80 Len=32

Frame 292: 164 bytes on wire (1312 bits), 164 bytes captured (1312 bits)
Ethernet II, Src: 40:00:00:02:00:01 (40:00:00:02:00:01), Dst: 40:00:00:02:00:02 (40:00:00:02:00:02)
Internet Protocol Version 4, Src: 10.2.0.2, Dst: 10.2.0.4
User Datagram Protocol, Src Port: 32984, Dst Port: 4789
Virtual eXtensible Local Area Network
Ethernet II, Src: 40:00:00:08:00:03 (40:00:00:08:00:03), Dst: 40:00:00:08:00:09 (40:00:00:08:00:09)
Internet Protocol Version 4, Src: 10.8.0.6, Dst: 10.8.0.12
User Datagram Protocol, Src Port: 2152, Dst Port: 2152
GPRS Tunneling Protocol
Internet Protocol Version 4, Src: 172.12.222.3, Dst: 172.12.222.254
User Datagram Protocol, Src Port: 10000, Dst Port: 80
Data (32 bytes)

0000  40 00 00 02 00 02 40 00  00 02 00 01 08 00 45 00   @·····@· ······E·
0010  00 96 85 c4 40 00 40 11  a0 89 0a 02 00 02 0a 02   ····@·@· ········
0020  00 04 80 d8 12 b5 00 82  00 00 08 00 00 00 00 09   ········ ········
0030  60 00 40 00 00 08 00 09  40 00 00 08 00 03 08 00   `·@····· @·······
0040  45 00 00 64 8d 3a 40 00  40 11 99 2d 0a 08 00 06   E··d·:@· @··-····
0050  0a 08 00 0c 08 68 08 68  00 50 bc 42 32 ff 00 40   ·····h·h ·P·B2··@
0060  00 00 00 03 00 2d 00 00  45 00 00 3c 2c cf 00 00   ·····-·· E··<,···
0070  57 11 21 c7 ac 0c de 03  ac 0c de fe 27 10 00 50   W·!····· ····'··P
0080  00 28 db 49 50 52 55 45  42 41 00 00 00 00 00 00   ·(·IPRUE BA······
0090  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ········ ········
00a0  00 00 00 00
```

Figure 16 - Snort IDS event against all overlay network information from the flow

As a novel capability, the NIDS should be able to detect simultaneously attacks being addressed over a 5G user, a tenant, or the entire infrastructure. The proposed architecture for the NFM component makes the following contributions and achievements to IoT 5G Network security:

- Distributed detection of the threat as an extension of traditional NIDS solutions for all network segments in a 4G/5G-IoT infrastructure.Support for encapsulation and tunnelling protocols inherent in 4G/5G deployments such as VXLAN, GRE, GTP and/or GENEVE.
- Extended North Bound Interface (NBI) providing IDS events to upper management layers providing fine-grained metadata about source and nature of the attack based on a DPI of the 4G/5G-IoT traffic.
- Protection awareness for digital service providers alongside the physical 5G infrastructure and also 5G user aware.

3.1.2.3 Produced resources

Due to the application of IPR safeguarding measures, UWS has not the intention to make any release of the source code for this component and thus it will not be made available in any public or private repository or server outside of our premises. For integration purposes, a functional prototype of the Network Flow Monitoring component will be released for the ARCADIAN-IoT consortium.

3.1.3 Design specification

3.1.3.1 Logical architecture view

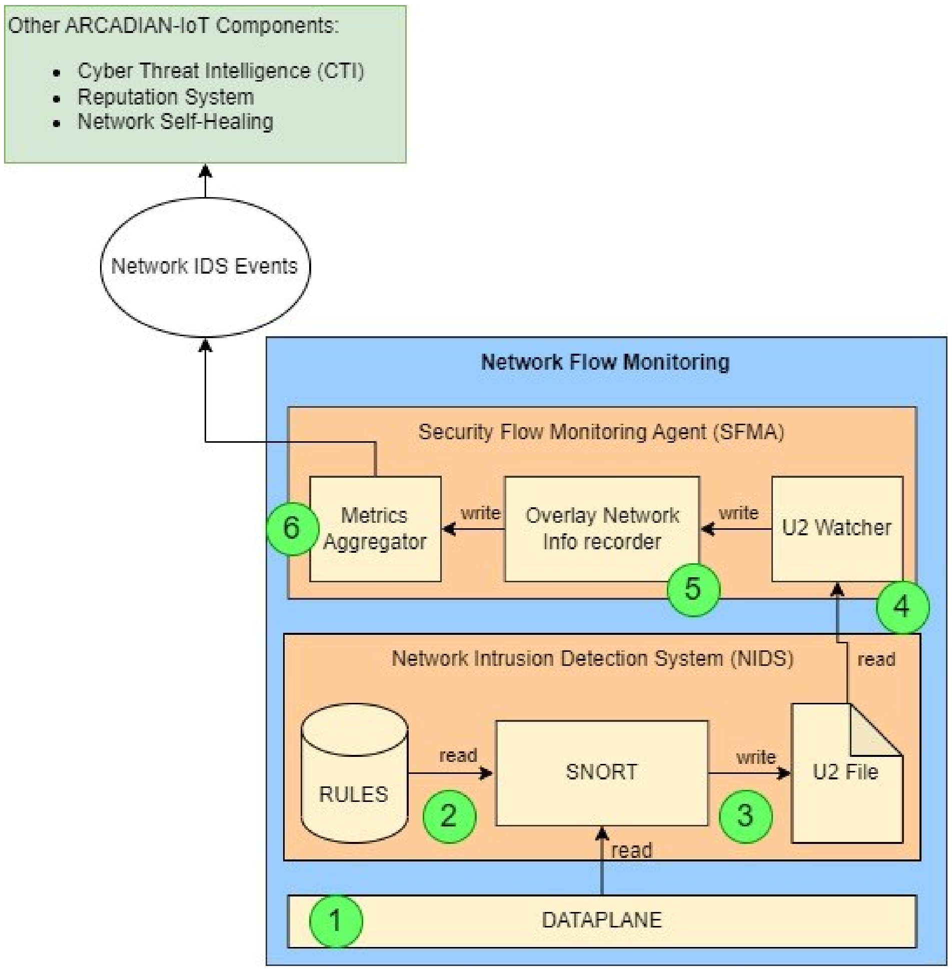

Other ARCADIAN-IoT Components:
Cyber Threat Intelligence (CTI)
Reputation System
Network Self-Healing
Network IDS Events
Network Flow Monitoring
Security Flow Monitoring Agent (SFMA)
Metrics Aggregator
write
Overlay Network Info recorder
write
U2 Watcher
6
5
4
Network Intrusion Detection System (NIDS)
read
RULES
read
SNORT
write
U2 File
2
3
read
DATAPLANE
1

Figure 17 - Logical architecture of the Network Flow Monitoring

Figure 17 describes the logical architecture that follows the Network Flow Monitoring component. The workflow of the component starts with the monitoring of the data plane in the network to which the NFM is attached (see 1 in Figure 17). A Network Intrusion Detection System (NIDS) is inspecting each packet in that data plane and will be executing a set of security rules (see 2 in Figure 17). When a rule matches with the packet inspected, the NIDS will write a new event in a Unified2 log file with information on the threat (see 3 in Figure 17). For this work it has been chosen Snort as NIDS and first subcomponent of the NFM.

The second and last subcomponent is the Security Flow Monitoring Agent (SFMA), whose workflow starts with the continuous reading of the Unified2 log file written by the NIDS. As soon as the SFMA U2 watcher collects a new event from the U2 log file, the SFMA will start working (see 4 in Figure 17). As the NIDS is not capable of recording full network flow information with all encapsulation information and so on, the SFMA is in charge of recording this overlay network information that lacks the NIDS (see 5 in Figure 17). Finally, the SFMA will aggregate all-important 5G-IoT overlay network information to the publisher exchange, building an Alert for the following network security self-protection control loop components (see 6 in Figure 17).

3.1.3.2 Sequence diagrams

Figure 18 describes the sequence diagram of the Network Flow Monitoring component in relationship with the consequent cognitive loop for network security component, the Network Self-Healing. The first loop oversees watching and reading all new events that the NIDS is writing continuously to the unified2 log file when detects a potentially malicious flow. Once this flow is detected, the Security Flow Monitoring Agent subcomponent will start the second loop, recording all nested overlay encapsulations belonging to the flow. Thus, the Metrics Aggregator can build the proper Alert with all fine-grain metadata information about the malicious flow detected and will send this created Alert to the Network IDS Events message bus exchange. Therefore, the Network Self-Healing will be able to perform its task with the information provided by the Network Flow Monitoring.

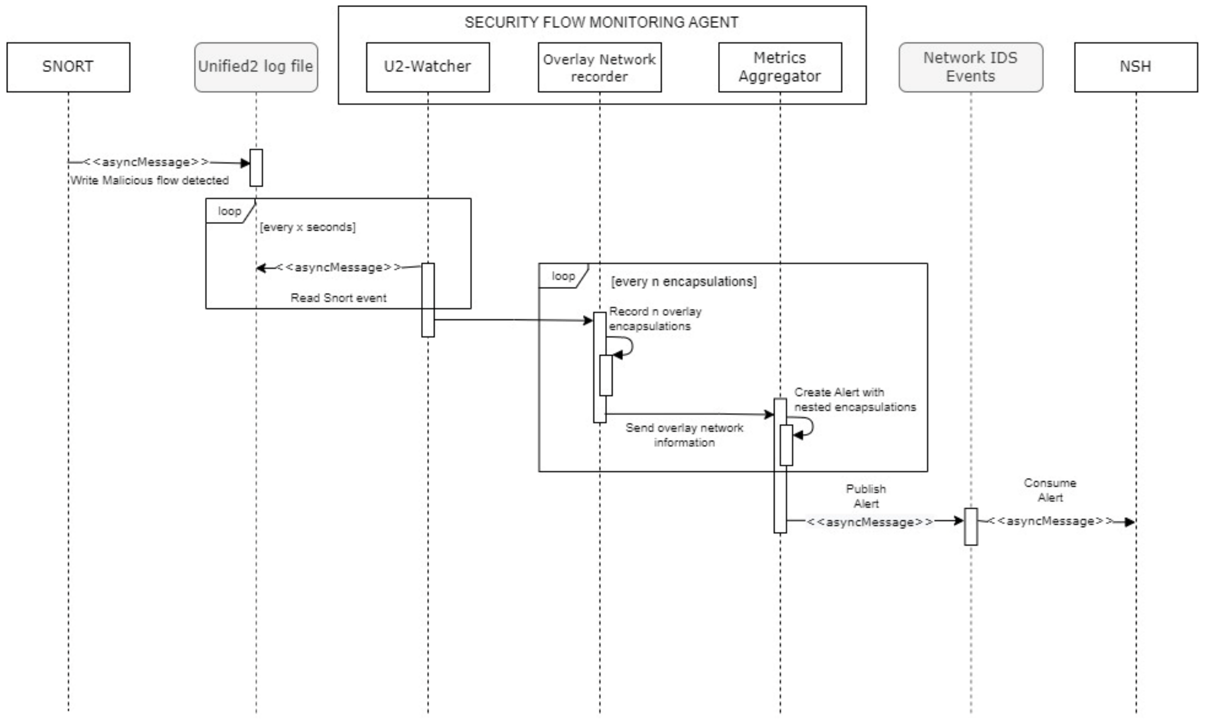


Figure 18 - Sequence diagram of the Network Flow Monitoring component

3.1.3.3 Interface description

- Network IDS Events is the interface where the Alerts from the Network Flow Monitoring will be published. All details and fields of these Alerts are described in previous Section 3.1.2.3.
- Network Self-Protection confirmations is the interface that the Network Self-Protection is using to provide information about the successful enforcement of the security rule provided by the Network Self-Healing component.

Partner	Definition	Channel	Exchange
UWS	Network Self-Healing	RabbitMQ AMQP 0.9.1	Network IDS Events
UWS	Network Self-Protection	RabbitMQ AMQP 0.9.1	Network Self-Protection Confirmations
UC	Reputation System	RabbitMQ AMQP 0.9.1	Network IDS Events
RISE	Cyber Threat Intelligence	RabbitMQ AMQP 0.9.1	Network IDS Events

Table 4 - External interfaces of the Network Flow Monitoring component

As a result of the development done on the Network Flow Monitoring component, a new resource has been produced. This is the JSON object that is being submitted to the Network IDS Events exchange for the message bus. The table below defines all fields of the JSON object.

Resource		
Resource is the JSON object with information about the malicious flow detected. This information is described in the table below and contains 5G and IoT network flow information.		
Name	**Description**	**Type**
encapsulationLayer	Specifies the number of encapsulation layers of the flow detected (0, 1 or 2)	Integer
encapsulationID1	Identifier of the first encapsulation layer level found in the detected flow (only when detected 1 or more encapsulation levels)	String
encapsulationID2	Identifier of the second encapsulation layer level found in the detected flow (only when detected 2 or more encapsulation levels)	String
encapsulationType1	Encapsulation name used in the first encapsulation layer level (i.e., gtp, vxlan, geneve...) (only when detected 1 or more encapsulation levels)	String
encapsulationType2	Encapsulation name used in the second encapsulation layer level (i.e., gtp, vxlan, geneve...) (only when detected 2 or more encapsulation levels)	String
sense	Sense of the flow that has been detected at the interface where the traffic is being mirrored, can take as value “INGRESS” or “EGRESS”	String
outMacSrc	Source MAC address of the detected flow	String
outMacDst	Destination MAC address of the detected flow	String
srcIP	Source IP address of the detected flow	String
dstIP	Destination IP address of the detected flow	String
outSrcIP	Outer source IP address of the consequent nested encapsulation of the detected flow	String
outDstIP	Outer destination IP address of the consequent nested encapsulation of the detected flow	String
l4Proto	Number code of the IP protocol	String
tos	Type of Service of the IPv4 header dataframe of the flow detected	String
srcPort	Source port number of the flow detected	String
dstPort	Destination port number of the flow detected	String

resourceId	Is the unique identifier of the Flow.	String
resourceType	Constant Value – Always "FLOW_SAMPLE"	String
state	Defines the actual state of the flow, it can take as value "ACTIVE" or "TERMINATED"	String
serviceInstanceResourceId	Is the unique identifier of the component that has reported the flow (useful for distributed NFM)	String
reportedTime	Is the number of milliseconds of the system when the NFM is reporting the malicious flow detected by the IDS	Long
Alert		
Alert is the JSON object with information about the alert reported to the rest of the system. This information is described in the table below and contains information about the type and importance of the alert.		
Name	**Description**	**Type**
alertType	Is the ordinal number of the classification configuration file of the IDS that matched with the IDS rule	String
alertReasonId	Is the IDS signature identifier	String
alertAssertionType	"NEGATIVE" or "INFORMATIVE"	String
alertImpact	Is the IDS severity of the malicious flow detected by the matched rule	Integer
alertTime	Is the number of milliseconds of the system when the Snort or other IDS has detected the malicious flow	Long
resourceId	Is the unique identifier of the alert	String
resourceType	Constant Value – Always "ALERT"	String
state	Is the actual state of the alert object in the system, can take as value: "FIRED" or "RETRACTED"	String
serviceInstanceResourceId	Is the unique identifier of the component that has reported the alert (useful for distributed NFM)	String
reportedTime	Is the number of milliseconds of the system when the NFM is reporting the alert	Long

3.1.3.4 Technical solution

3.1.3.4.1 Deployment architecture view

Figure 19 denotes the global deployment architecture view followed by the UWS cognitive loop for network security components alongside an entire 5G network infrastructure. The 5G architecture is composed of different network segments and layers. It can be differentiated the Radio Access Network (RAN) segment, where IoT devices are connecting to the different 5G gNB with its respective Distributed Unit (DU) that connects to the consequent network segment, the Edge. This network segment oversees the responsibility of deploying and managing the digital services from the rest of the stakeholders involved in the system, as Digital Communication Services Providers. They use their virtual Central Units to manage communications across the rest of the infrastructure. In addition, in the Edge segment can be found Multi-access Edge Computing applications in order to approach computing services and capabilities to the end user. The transport network segment is the one between the Edge and Core premises, where the traffic is routed from the specific geographically distributed Edge to the centralized Core. The last network segment in the infrastructure is the Core, where first user authentication takes place. Once the communications leave the Core segment, they are forwarded to an inter-domain network segment outside the infrastructure provider's domain.

The Network Flow Monitoring component has two different deployment views. A first view is a centralized approach in a Service Layer where traffic across the network needs to be mirrored. The second view is a distributed approach, where the NFM can be instantiated alongside the different segments and stakeholders of the network, performing distributed and multi-layered detection of potentially dangerous threats.

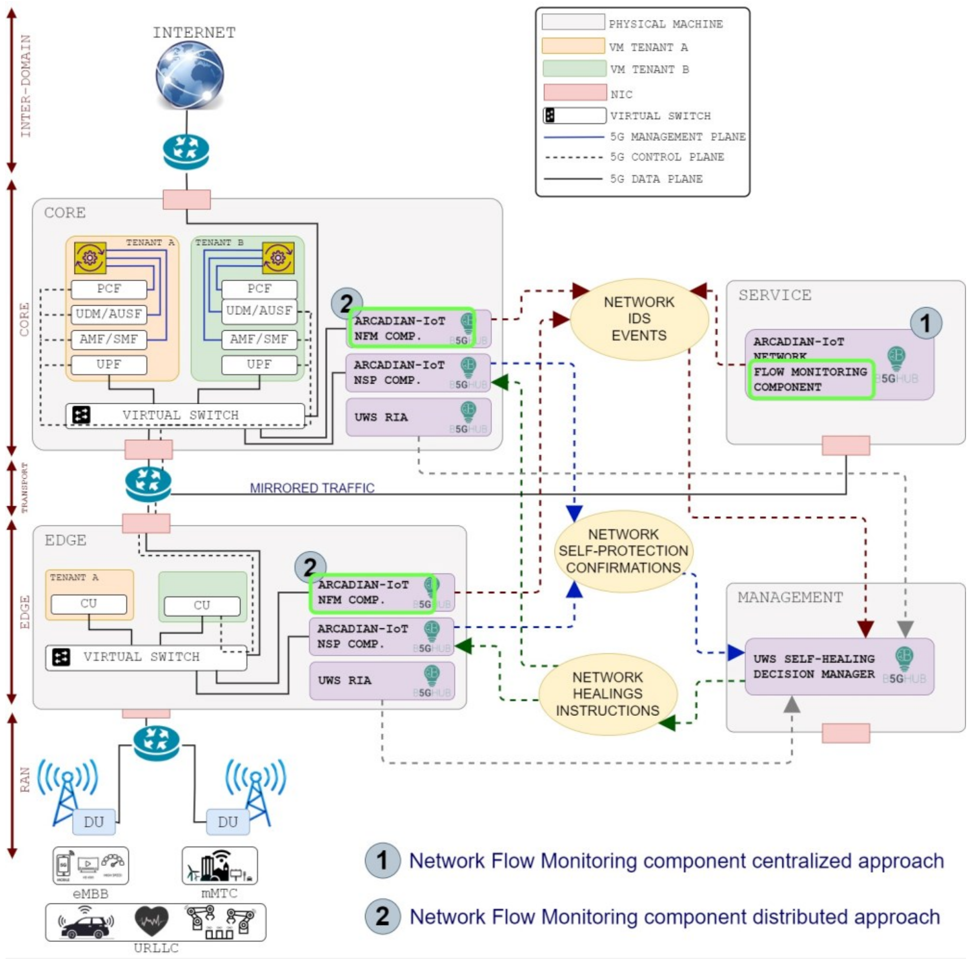


Figure 19 - Deployment architecture view of the Network Flow Monitoring component

3.1.3.4.2 API specification

Network Flow Monitoring component is using a message bus publication/subscription channel to interact with the rest of the components in the ARCADIAN-IoT framework. This component uses two different exchanges: the first is the Network IDS Events, where the Network Self-Healing, Cyber Threat Intelligence and Reputation System will be retrieving Alerts. As a second interface, the NFM component will be subscribed to the Network Self-Protection confirmations. This information will be used to confirm the security rule has been already enforced by the NSP component and thus the effectiveness of the cognitive self-protection security network loop will be also confirmed.

3.1.4 Evaluation and results

The testbed used for the empirical validation and evaluation is the same for the 3 ARCADIAN-IoT components composing the Network Self-Protection Loop, that is, the Network Flow Monitoring, the Network Self-Healing, and the Network Self-Protection. These 3 components set the self-protection loop provided by the UWS which, the NFM detects the malicious flows; the NSH analyses and creates the healing plans; and the NSP mitigates the incoming attack. Thus, in the following sections regarding the NSH and the NSP, the testbed description, the different scenarios described in Table 4 and the results figure in this section will be referenced.

- **Framework Validation:** To substantiate the functionality and practicality of the proposed framework in this research, a dedicated testbed environment was meticulously developed. This testbed served as the platform for conducting an extensive array of experiments aimed at collecting data for subsequent analysis.
- **Testbed Configuration:** The testbed was designed to emulate a 5G network infrastructure closely resembling the configuration illustrated in Figure 19 of the research. In this emulation, the network infrastructure was subjected to a UDP-based Distributed Denial of Service (DDoS) attack, originating from compromised IoT devices.
- **Scenario A:** The empirical validation of the proposed framework was conducted in two distinct scenarios. In the first scenario (A), the focus was on manipulating the number of attackers, with varying quantities of User Equipment (UEs) connected to each gNB. The remaining network elements in this topology remained constant, comprising one gNB linked to each Data Service Provider (DSP), two DSPs per Internet Service Provider (ISP) with multi-tenant isolation, and two Edge nodes interfacing with one Core node.
- **Scenario B:** In the second scenario (B), a more intricate network topology was established to expand the potential points of action for mitigating the attack. While maintaining the same number of attackers as in Scenario A, the network topology was configured as follows: two gNBs connected to each DSP, two DSPs per ISP, and four Edge nodes interfacing with one Core node.

These scenarios were devised to assess the performance and efficacy of the proposed framework under different conditions, allowing for a comprehensive evaluation of its capabilities in handling various network configurations and attack scenarios, thus validating the proposed KPIs.

Scenario	A				B			
No UE x gNB	4	8	12	16	1	2	3	4
No gNB x DSP	1	1	1	1	2	2	2	2
No DSP	2	2	2	2	2	2	2	2
No Edge	2	2	2	2	4	4	4	4
No Core	1	1	1	1	1	1	1	1
Total attackers	16	32	48	64	16	32	48	64
Packet rate each UE (pps)	5600	2800	1875	1400	5600	2800	1875	1400
Consumed BW each UE (Mbps)	64	32	22	16	64	32	22	16

Table 5 - Configuration of 8 different experiments for Scenarios A and B

The research encompasses a series of experiments, as detailed in Table 5, with results graphically presented in Figure 20, divided into two sections representing scenarios A and B. In both sections, the X-axis quantifies the number of infected IoT devices (16, 32, 48, and 64), while the Y-axis portrays the average time required by the Cognitive Self-protection framework for the complete process from attack detection to mitigation, measured in seconds. The graph employs distinct colours to signify the time consumption by each framework component: blue for Network Flow Monitoring (NFM), orange for Network Self-healing (NSH), and green for Network Self-protection (NSP). Results for NSP and NSH are discussed in sections 3.4 and 3.6 respectively. This visual representation offers insights into the efficiency of the framework in scenarios A and B, showcasing variations in response times with varying numbers of infected IoT devices.

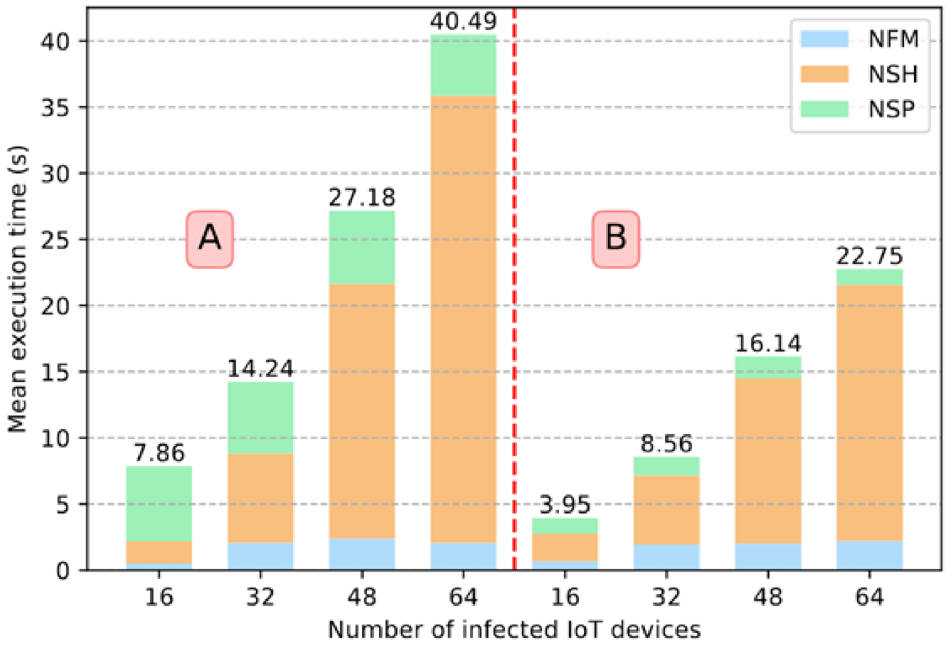


Figure 20 Average time consumed by the Cognitive Self-protection loop, Scenario A (left) against Scenario B (right)

Regarding the individual impact of each component, the Network Flow Monitoring (NFM) component exhibits noteworthy performance superiority compared to the other two main actors involved in the self-protection loop (i.e. NSH and NSP). Hence, it is worth noting that the NFM component does not add relevant load to the system. Its performance remains consistently stable across all scenarios (A and B), even as the number of infected IoT devices increases. In the most challenging scenario involving 64 IoT devices, NFM demonstrates efficient response times, consuming 2.06 seconds in scenario A and 2.20 seconds in scenario B. These findings underscore the promising scalability of the NFM regarding the network topology size, the number of IoT UEs simultaneously connected and the number of different flows handled by this ARCADIAN-IoT architectural component.

Furthermore, it is essential to emphasize that all the configurations outlined in Table 4 encompass scenarios and topologies characterized by the involvement of multiple stakeholders, the incorporation of diverse network segments, and the utilization of various overlay network layers within the network architecture. Consequently, the Key Performance Indicators (KPIs) for the Network Flow Monitoring (NFM) component have been comprehensively addressed and successfully attained across these experimental scenarios.

3.1.5 Future work

Integration with ARCADIAN-IoT Framework Components: Future work should focus on the seamless integration of the proposed framework with other components of the ARCADIAN-IoT framework, particularly the Cyber Threat Intelligence (CTI) and

Remediation System (RS) components. This integration would enhance the overall cybersecurity ecosystem.
Beyond ARCADIAN-IoT - Multi-Interface NIDS: A prospective avenue for research involves the implementation of multiple Network Intrusion Detection System (NIDS) instances to concurrently monitor and assess multiple network interfaces. This expansion would bolster the framework's ability to comprehensively monitor and protect diverse network segments.
Beyond ARCADIAN-IoT - Enhanced NFM Capabilities: Further research should explore the augmentation of Network Flow Monitoring (NFM) capabilities to address emerging network-based cyberthreats. Specifically, investigating threats such as DNS and ARP spoofing would extend the NFM's capacity to identify and mitigate a broader spectrum of cybersecurity risks.

3.2 Device Behaviour Monitoring

3.2.1 Overview

3.2.1.1 Description

One of the ARCADIAN-IoT project’s objectives was to develop a component that detects anomalous behaviours on devices. The developed component is a host-based intrusion detection system (HIDS) directed for IoT devices. The component takes the task of examining events that are specific to the host device (e.g., device reputation, sequences of system calls or authentication attempts).

The Device Behaviour Monitoring (DBM) component is comprised by a set of subcomponents that collect and classify the events with the use of Machine Learning (ML) models. The models are updated via a FL scheme that increases privacy preservation of the devices.

The anomaly detection is performed in real time to make sure that the device is operating normally and detect signs of anomalies upon abnormal operational behaviour (e.g., malfunctioning or intrusion attempts). The resulting output of this component consists of a value that indicates whether the event was classified as an anomaly (i.e., the activity relates to an ongoing anomaly or intrusion attempt) or normal behaviour, with a respective confidence level associated with the classification. The resulting information is then encapsulated and forwarded to other ARCADIAN-IoT components via a message bus (RabbitMQ) or in-device communication (inter-thread communication). Specifically, an anomaly is detected, the information should be sent, for instance, to the IoT Device Self-protection component, which has the role of protecting devices against anomalous behaviours or incoming attacks. In a complementary way, there were applied two eXplainable AI (XAI) methods to retrieve explanations for the component's results and provide insights about the impact of certain system calls. Further interfaces are described later.

3.2.1.2 Requirements

A recall of the high-level requirements that have been previously defined and provided in deliverable D2.4 is given below.

Requirement 3.2.1 – User logs access: The Behaviour Monitoring component must have access to device’s user logs in order to detect possible security issues.

Requirement 3.2.2 – Device permissions: The Behaviour Monitoring component must be aware of permissions granted to applications/services accessing the device.

Requirement 3.2.3 – Local model training capabilities: The ML models should be lightweight and be able to run on the device.

Requirement 3.2.4 – Response to anomalies: The Behaviour Monitoring component should be able to send an alarm, in real time, when anomalous behaviour is detected.

Requirement 3.2.5 – Secure Communication: The Behaviour Monitoring component communications between the devices and the central server should be secured (e.g., using encryption).

3.2.1.3 Objectives and KPIs

A recall (with supplemental information) of the objectives and KPIs defined in the agreement is provided bellow.

KPI scope	
Applicability of IDS models in IoT devices	
Measurable Indicator	
Model performance (e.g., TPR)	
Benchmarking	
Accuracy >= 0.96 TPR >= 0.95 FPR <=0.05 F1-Score >= 0.96 [38]	
Target value	***Achieved value***
Accuracy >= 0.97 TPR >= 0.98 FPR <=0.01 F1-Score >= 0.99	Accuracy ~= 0.97 TPR >= 0.98 FPR <=0.01 F1-Score >= 0.99

KPI scope	
Ability to process different input types with non-root privileges (e.g., syscalls, auth, rep)	
Measurable Indicator	
Number of inputs considered	
Target value	***Achieved value***
Consider at least 3 types of input (e.g., syscall sequences, reputation score, authentication results)	Considers 3 types of input (I.e., syscall sequences, reputation score, authentication events)

KPI scope	
Deployment in heterogenous devices	
Measurable Indicator	
Number of supported devices	
Target value	***Achieved value***
Support at least 2 types of devices (e.g., drone device, smartphone, industrial IoT device – developed by BOX2M)	Supports 2 types of devices (drone device and smartphone)

3.2.2 Technology research

This section provides a background on the topic of the solution, presents the research done relative to the current state of the art of Host Intrusion Detection solutions, research findings as well as achievements and produced resources.

3.2.2.1 Background

Intrusion Detection based on System Call Anomaly Detection

The main objective of an Intrusion Detection System (IDS) is to examine activities within a system or a network, in order to identify possible intrusions from malicious sources. Generally, IDS can be divided into two main groups, Host Intrusion Detection System (HIDS) or Network Intrusion Detection System (NIDS). This component is part of the former group (i.e., HIDS).

Furthermore, intrusion detection can be performed based on two different approaches: Signature Intrusion Detection system (SIDS) or Anomaly Intrusion Detection System (AIDS). A brief comparison between the two techniques is summarised in Table 6 - Comparison between SIDS and AIDS. The first (SIDS) relies on a system which can be based on known threats (with rules about the collected data), while the second (AIDS) monitors the system behaviour to detect abnormalities that deviate from the normal baseline behaviour. Both methods have their benefits, with signature detection being accurate and working well on known threats, but being unable to detect zero-day attacks, contrary to AIDS. On the other hand, anomaly detection requires characterizing and identifying the complete normal behaviour of the system, which can be difficult to achieve.

	Advantages	Disadvantages
SIDS	• Very effective in detecting intrusions with a very low false positive rate; • Promptly identifies the intrusions; • Superior in detecting already known attacks; • Simpler design.	• Needs to be updated frequently with new signatures; • If an existing known signature (i.e., intrusion) suffers from slight deviations, the system wouldn't be able to detect that signature; • Is not able to detect zero-day attacks; • Not suited to detect multistep attacks.
AIDS	• Can be used to detect unknown attacks; • Could be used in order to create an intrusion signature.	• Has a high false positive rate; • Hard to build a normal profile for a dynamic computer system; • Unclassified alerts; • Needs initial training.

Table 6 - Comparison between SIDS and AIDS

In HIDS, the data from the host system's audit and logging mechanisms are analysed to look for signs that the system has been broken into or a possible attack. The data may include activity from the local hosts' logins, file access, privilege escalation, alteration of system privileges, system calls (the focus of this component), file system changes and application logs, and many others.

Consider a scenario where an IoT device is being used on an open network and a third party intends to perform malicious actions on the device. To detect these malicious actions, one possibility is to search for specific characteristics of the attack. For instance, the attacker performs actions/commands on the system that are different from the way a typical user would. This can be used to create a detection system that will search for these deviations from the normal behaviour on the IoT device. Some examples of these anomalies are unusual read/write (or other system calls) behaviour or multiple login attempts in a certain timeframe to the machine.

System call traces are often used in detecting intrusions with HIDS on program level. System call traces are used to find repeated patterns of system calls, enabling anomaly detection and misuse detection during execution, intrusions could be in the form of sub-sequential traces of intrusive activities. A system call is a fundamental interface between a program on a machine and the operating system. The system call is a way for a user program to request an operation to be completed by the operating system.

System calls are an adequate data source for HIDSs because they are a primary artifact of the OS kernel, and their collection imposes low computational overhead. The effective use of system calls within host-based anomaly detection was first proposed by Forest et al.[39] with the following advantages:

- System calls (i.e., root processes) are more advantageous than a typical user process because a system call will have greater access to the system's resources.
- An operating system will have a finite list of operations, which creates predictable system call sequences under normal system operations.

Forrest et al. processed short sequences of system calls to generate profiles of normal program behaviour. This approach is based on an enumeration sequence-based method known as STIDE (Sequence Time Delay Embedding). In this technique, the sliding window method is used to generate short sequences of system call traces and then a database with the signature of the normal behaviour with the invoked sequences. This algorithm was inspired by the natural immune systems of organisms. The approach is claimed to be simple and efficient to deploy for possible real-time implementation.

Lee et al. [40] attempted to improve the results obtained by Forrest by using machine learning algorithms to extract information from normal and abnormal sequences of system call traces. Frequency based methods were also explored by Helman and Bhangoo [41]. However, instead of keeping track of frequencies of each system call individually, the authors recorded frequencies of sequences of system calls by tokenizing the system calls into *ngram* sequences, which enabled them to preserve the order of the system calls.

Liao and Vemuri [42] took a slightly different approach. Instead of keeping track of sequences of calls, they monitor the frequency of system calls that are issued by a program. By doing this, they avoid having to treat every system call individually and reduce the overhead involved in analysing and storing every system call. A system call is considered an instance of a text and the whole set of calls issued as a *document*.

In traditional Machine Learning (ML) approaches, it is common to have a centralized server or a cloud platform, where data from several devices (e.g., smartphones, tablets, laptops, smart carts) is aggregated and consequently used to train the central model. Such devices collect vast amounts of data, often sensitive or private, which inherently creates a significant risk by having such data stored on a central server. Thus, it is essential to employ solutions that help maintain user data private and secure. Back in 2016, as part of an effort to devise a decentralized method for using data from mobile devices to improve the user experience of other AI focused solutions, Google proposed Federated Learning. In Federated Learning, the local training data is kept on device-level, meaning that the client data is not directly transmitted over the network to prevent any data from leaking, therefore ensuring a basic level of user or device privacy. The only data to ever be transferred are the locally computed updates from each device to a central aggregator server, which is a cloud-based distributed service (referred to as a server model) where data are aggregated.

Overall, the behaviour monitoring component employs a HIDS approach complemented by a Federated Learning design to train the MLP-based model.

Authentication and Reputation based Anomaly Detection

The DBM component uses additional data to further monitor the behaviour of the device, this additional information is composed of time series data that comes from authentication logs and reputation score changes over time.

In the analysis process of such data, we aim to detect anomalies in said data, using anomaly detection techniques. Anomalies are data instances that have deviating characteristics from the baseline behaviour patterns. Most existing model-based approaches to anomaly detection construct a profile of normal instances, then classify data points that deviate from the normal profile as anomalies. There are many notable examples of ML based methods such as supervised methods, semi-supervised methods, and unsupervised methods.

The most appropriate method that helps the implementation of a completely autonomous anomaly detection system based on authentication inputs is the unsupervised method, which enables us to detect anomalies without having to train a machine learning model *a priori*. As for the reputation inputs, the component employs a crisp classification method based on different levels of reputation.

One such algorithm that detects outliers is Isolation Forest [43]. Isolation Forest is a popular unsupervised learning algorithm that identifies anomaly by isolating outliers in the data and is based on the Decision Tree algorithm.

In general, the first step to anomaly detection/unsupervised learning is to construct a profile of what's the baseline behaviour, and then detect any instances that are not considered normal as anomalous. However, the Isolation Forest algorithm does not work on this principle, it does not first define "normal" behaviour, and it does not calculate point-based distances, which is why it is not necessary to pre-train the model with the baseline behaviour. Instead, it works by explicitly isolating anomalous points in the dataset. The main presumption in Isolation Forest algorithm is based on the principle that anomalies are observations that are few and different from the rest of most data points. Thus, Isolation Forest uses an ensemble of Isolation Trees for the given data points to isolate anomalies.

eXplainable Artificial Intelligence (XAI)

There are many factors that could contribute to how a model operates and makes its predictions and, therefore, many ways to explain it. The XAI methods can be classified using two dimensions: stage and scope. An explainability method can be applied through the several stages of an AI development model. The scope of the explainability method can either be global, where the method provides an explanation global with respect to the model, or one that is local, with respect to a prediction. There are two main categories of model approaches to explainability: intrinsic (modelling explainability) and post-hoc (post-modelling explainability) [51]. Intrinsic models are interpretable models such as decision tree or rules. Post-hoc is related to extracting the information from already taught models and it does not precisely depend on how the model works - hence it is model-agnostic, or if applied on a Deep Learning (DL) model, model specific.

Although XAI is a research field with substantial active research, it is mainly focused on fields like healthcare, natural language processing, computer vision, etc. The interpretability methods are seldomly used for intrusion detection. Nevertheless, despite intrusion detection systems becoming more complex, some works to improve the interpretability of such models have been already produced. For instance, M. Wang et al. [44] proposed a framework that leads to improve the transparency of any IDS, presenting the first use of SHAP to provide local and global explanations for intrusion detection.

S. Patil et al. [45] proposed an innovative intrusion detection system, using ensemble methods of machine learning and incorporating LIME (i.e., a XAI method) for better explainability and understanding of the black-box approach to reliable intrusion detection. The experimental results confirm LIME is more explanation-friendly and more responsive.

D. L. Marino et al. [46] presented an approach to generate explanations for incorrect classifications made by data-driven IDS. An adversarial approach is used to find the minimum modifications (of the input features) required to correctly classify a given set of misclassified samples.

S. Mane et al. [47] have used deep neural network for network intrusion detection and also proposed explainable AI framework to add transparency at every stage of machine learning pipeline. The explanations are generated from SHAP, LIME, Contrastive Explanations Method (CEM), ProtoDash and Boolean Decision Rules via Column Generation (BRCG). The approaches were applied to NSL-KDD dataset for intrusion detection system (IDS).

Even though previous work uses XAI methods for intrusion detection, it focuses on the part of providing explanations. On the other hand, we went further and explored the explanations, analysing and validating them, obtaining concrete results on the importance of different features for different scenarios.

3.2.2.2 Research findings and achievements

System Call Intrusion Detection

The approach followed for the development of the DBM component has started with the validation of Machine Learning (ML) models on an open-source system call dataset known as ADFA-LD [48]. This dataset was used to validate ML classifiers and consequently the approach chosen for the classification of system call sequences. The results obtained with that dataset are demonstrated in the following sections.

Our approach to train and test models for Host Intrusion Detection System (HIDS) relies on system log events, in this case system calls. Since we are developing an anomaly detection system, training data must contain baseline and attack data. Two approaches were evaluated and compared: a centralized and a federated approach.

The proposed training architectures are a centralized model and a decentralized FL-based approach, where each of the nodes represented IoT clients. The main idea of this approach is to establish a comparison between the traditional IDS approaches and the federated setting, where the centralized architecture serves as baseline.

The applied methodology for the development of the IDS is divided mainly into two parts. The first one is an *offline* (i.e., non-deployed) approach. In this case, the goal is to validate the use of ML classifiers in the process of intrusion detection using system call sequences.

On the second part, we collected and aggregated the data from emulated devices, processed it and then trained the models with said data, with both normal and abnormal device behaviours. Regarding performance indicators of accuracy, F1-score, false positive rate, and true positive rate, the models that perform better are the Random Forest and the Multilayer Perceptron (MLP) as seen in Table 8. However, we have opted for the MLP for two reasons. Firstly, considering the size, the MLP stands out as the preferred choice due to its lightweight nature compared to the other models. Additionally, the challenge of aggregating parameters from traditional machine learning classifiers, such as the Random Forest, prevents their use in an FL-based approach. We then proceeded to develop the system that functions in *online* (i.e., deployment) mode as a system that is continuously monitoring the device. The detailed architecture of the system is depicted in Figure 21.

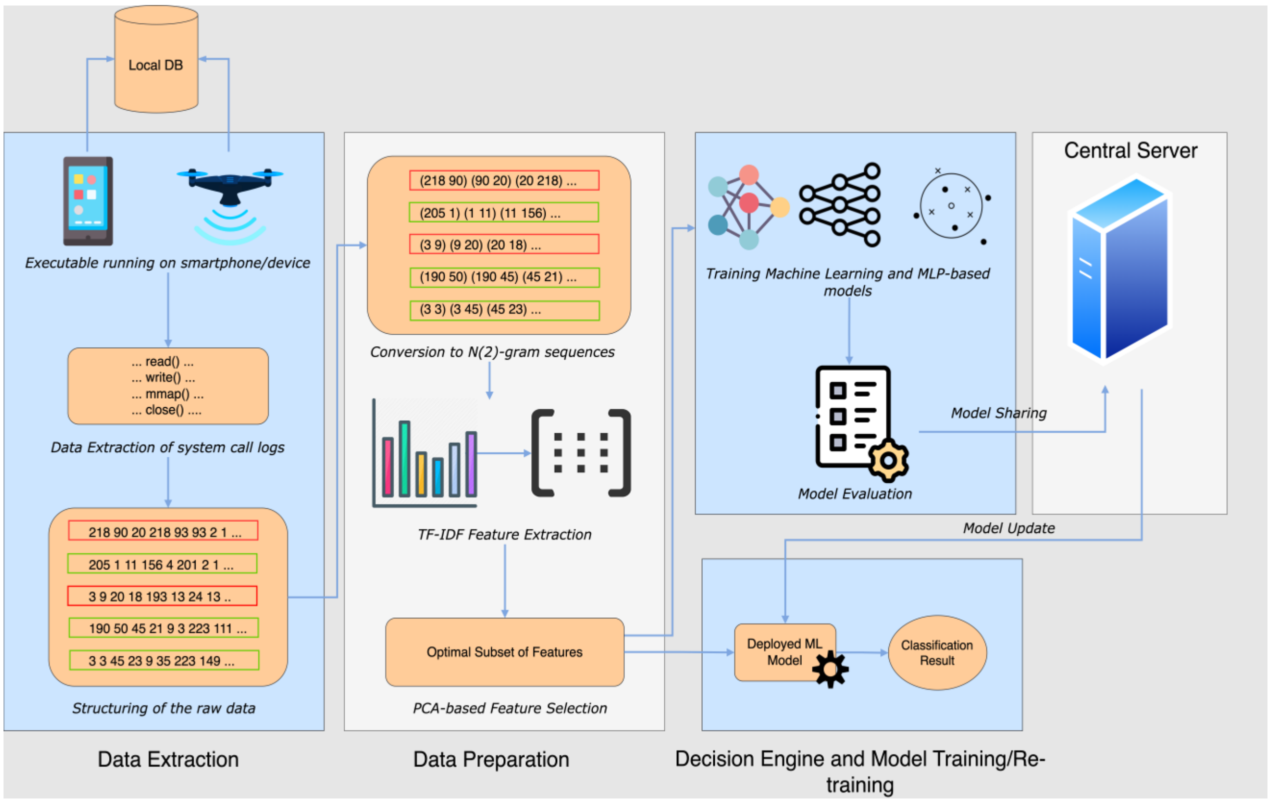


Figure 21 - Detailed architecture of the system call-based HIDS solution

The starting point of the system is the data extraction/tracer module, typically deployed directly on the devices (in edge cases the system calls can be relayed to this component when the deployment directly on the device is not feasible). This module collects and aggregates the system call traces, which are going to be processed and then are going to be the subject of automated analysis by a ML model unit that classifies the input as anomalies or normal, and then raises an alert when an anomaly is detected.

The proposed training architectures are a centralized model and a decentralized FL-based approach, where each of the nodes represent IoT clients. The main idea of this approach was to establish a comparison between the traditional IDS approaches and the federated setting, where the centralized architecture served as baseline.

The considered approach is based on continuous analysis of system call flows in devices, by capture of dynamic properties of processes running on the system. Thus, system call traces of running processes are extracted and analysed. The use of system call sequences for generating models that detect normal and anomalous sequences is justified by the fact that security violations on device level are likely to produce abnormal system call invocations. System calls are the only means by which a program operating in user space can enter kernel space and make use of the services provided by the kernel. The user space processes running on the Operating System (OS) make system calls to request services from the kernel.

In the last step, the system call sequences are generated. Each line of the recorded trace file represents a log related to a system call, process name, process ID, name of system call, related parameters, and return value. The system call names and associated process ID were extracted from each of the logs, and converted into system call numbers by referring to the system call table for the specific Linux version that was used, as each system call has a distinct numeric reference. The resulting data that was extracted is composed of system calls, each executed system call per line. These sequences need to be treated and processed to have a structured dataset that can be used for training Machine Learning models. In this step, we first tokenize the system calls into sequences of variable length.

For instance, considering the following snapshot of system calls recorded on a normal setting:

open(), mmap(), read(), socket(), mmap(), execve(), open(), read(), close(), brk()

With a sequence size of 5, the following subsequences would be produced by the sliding window method of size 1:

- Sequence 1: open(), mmap(), read(), socket(), mmap()
- Sequence 2: mmap(), read(), socket(), mmap(), execve()
- Sequence 3: read(), socket(), mmap(), execve(), open()
- Sequence 4: socket(), mmap(), execve(), open(), read()
- Sequence 5: mmap(), execve(), open(), read(), close()
- Sequence 6: execve(), open(), read(), close(), brk()

After the tokenization process, the data is sanitized by removing the intersecting rows (sequences) from both normal and attack datasets to maximize the distinction between the two class types for the learning process. A row with a normal sequence is set with a label of 0, whereas a row with an anomalous sequence is set with a label of 1. This is followed by a feature extraction process. This process can be done through various other approaches, in this work we evaluated the following three approaches: Trivial Representation, Vector Space Model and Term Frequency–Inverse Document Frequency. The Feature Extraction step results in 2805 features, represented as n-grams (tuple of n words where n is 2). After performing the Feature Extraction step, the 150 most relevant features are selected through a PCA-based Feature Selection. The resulting dataset is then divided into a 70/30 split for training and testing, respectively.

Authentication Log Analyser

The Device Behaviour Monitoring component also considers failed authentication attempts associated to the current device (matched via device ID).

It considers the timestamps of the failed authentication attempts and process their timestamp values in consecutive, sliding windows of size 10, in which time differences between two consecutive failed authentication timestamps and corresponding mean and standard deviation are calculated within that window. The means and standard deviations for each of the windows are then analysed by an Isolation Forest model, to detect anomalous authentication events, possibly linked to cyberattacks. If such events are detected, is sends a threat alert to the respective external interfaces (see section 3.2.3.3) for further actions.

Reputation Score Analyser

The device Behaviour Monitoring component keeps the history of the reputation score changes over time, and reports when a sudden change in reputation happens via change point detection. For reference, change point detection detects the data points where the underlying properties, such as statistical characteristics (e.g., mean and variance) of the time series shift abruptly (Figure 22 shows an example of data point jumps detection).

Reputation Scores Usage in DBM

Reputation scores of the device are also considered in various processes of the Device Behaviour Monitoring component. This is reflected in the decision process of sending the detected anomalies in collected system calls sequences based on the current reputation score of the device. In other words, when an intrusion is detected via system call analysis (and only for system calls anomalies), the component will decide if it will send the alert via the RabbitMQ based on the latest reputation score update and the confidence score given by the Machine Learning model. For example, an alert is sent in the following conditions:

- Confidence level is at least 0.9 and device score is either at most 0.5 or at least 0.9.
- Confidence level is at least 0.7 and device score is at most 0.3.

- Confidence level is at least 0.95, regardless of the device score.

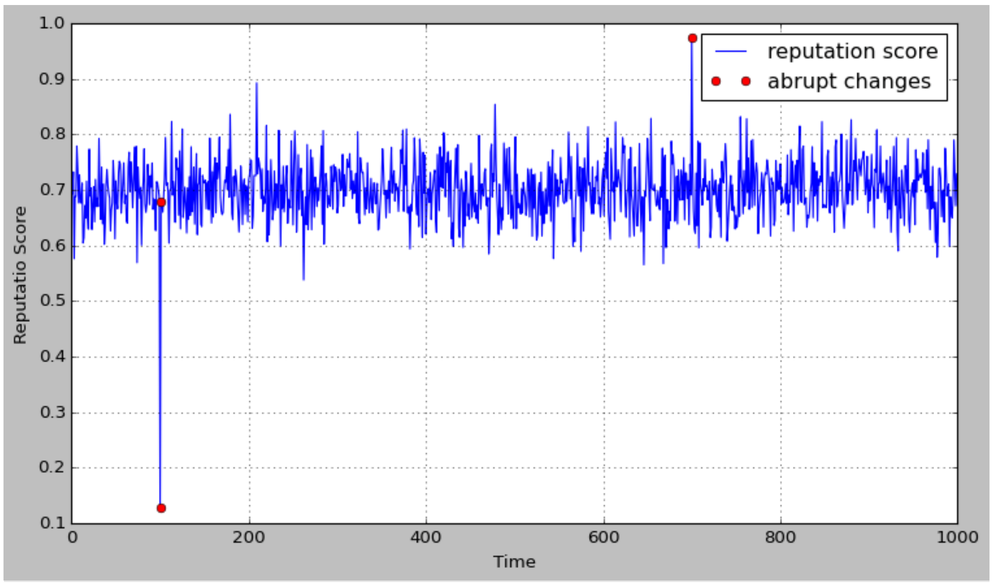


Figure 22 - Example of change detection

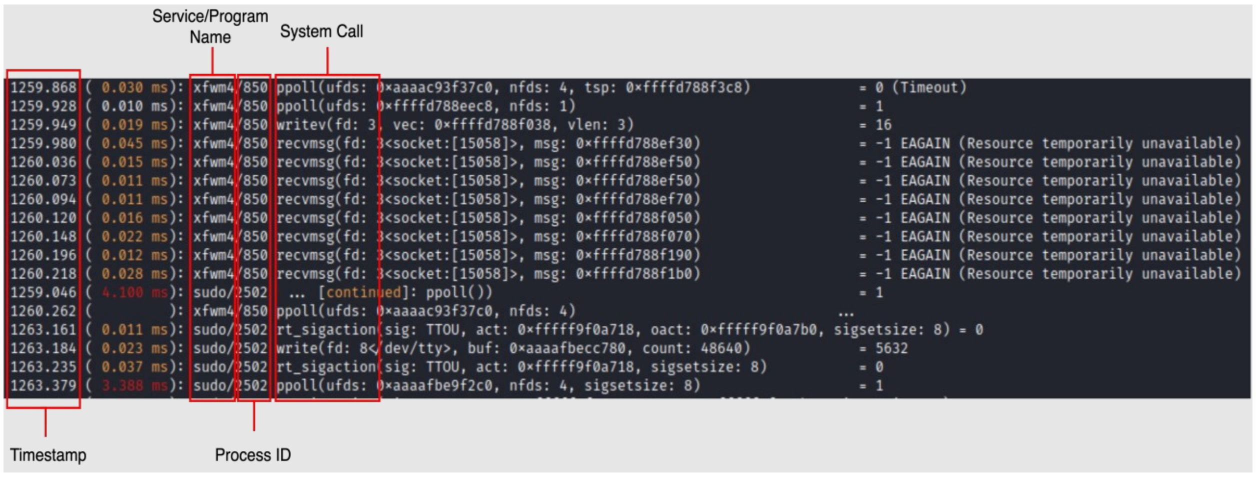
Model Explainability

In order to address the model explainability, two XAI methods were applied: LIME and SHAP. The methods are model-agnostic and enable local explanations. For every instance of the dataset, explanations were provided using both methods. With the explanations, the 10 most important system calls for each instance classification were retrieved. This is, if the classification of a particular instance was 0 (normal), LIME and SHAP would provide the 10 most important system calls for that classification, present in the input sequence. It is noticeable that LIME and SHAP might have different results. Based on these results retrieve system calls that are present only in the explanations of particular cases (for instance system call "x" being present only in cases where the result is a true positive). With the explanations provided by these methods, it becomes easier/faster to interpret the results of the model. In order to evaluate and validate the results of the XAI methods, perturbation analysis was carried out. With it, we were able to obtain results about the impact of different system calls on different scenarios. All these results can be observed in the next sections.

3.2.2.3 Produced resources

The following functionalities are the result of the research and development performed over the Device Behaviour Monitoring component:

- Data Extraction Module: The starting point is the extraction of device logs in real time. This module extracts system call logs with the *perf* Linux tool [49] of the host device in real time. Figure 23 presents an example of a raw system call log extracted with the *perf* tool (it is also possible to use strace). The logs are parsed and the relevant information from the log is extracted, such as the invoked system call, the process ID and the service which invoked the system call. The extracted information from each log is stored in a local database, with the data being processed by the following subcomponents.



```
1259.868 ( 0.030 ms): xfwm4/850 ppoll(ufds: 0×aaaac93f37c0, nfds: 4, tsp: 0×ffffd788f3c8)            = 0 (Timeout)
1259.928 ( 0.010 ms): xfwm4/850 ppoll(ufds: 0×ffffd788eec8, nfds: 1)                                 = 1
1259.949 ( 0.019 ms): xfwm4/850 writev(fd: 3, vec: 0×ffffd788f038, vlen: 3)                          = 16
1259.980 ( 0.045 ms): xfwm4/850 recvmsg(fd: 3<socket:[15058]>, msg: 0×ffffd788ef30)                  = -1 EAGAIN (Resource temporarily unavailable)
1260.036 ( 0.015 ms): xfwm4/850 recvmsg(fd: 3<socket:[15058]>, msg: 0×ffffd788ef50)                  = -1 EAGAIN (Resource temporarily unavailable)
1260.073 ( 0.011 ms): xfwm4/850 recvmsg(fd: 3<socket:[15058]>, msg: 0×ffffd788ef50)                  = -1 EAGAIN (Resource temporarily unavailable)
1260.094 ( 0.011 ms): xfwm4/850 recvmsg(fd: 3<socket:[15058]>, msg: 0×ffffd788ef70)                  = -1 EAGAIN (Resource temporarily unavailable)
1260.120 ( 0.016 ms): xfwm4/850 recvmsg(fd: 3<socket:[15058]>, msg: 0×ffffd788f050)                  = -1 EAGAIN (Resource temporarily unavailable)
1260.148 ( 0.022 ms): xfwm4/850 recvmsg(fd: 3<socket:[15058]>, msg: 0×ffffd788f070)                  = -1 EAGAIN (Resource temporarily unavailable)
1260.196 ( 0.012 ms): xfwm4/850 recvmsg(fd: 3<socket:[15058]>, msg: 0×ffffd788f190)                  = -1 EAGAIN (Resource temporarily unavailable)
1260.218 ( 0.028 ms): xfwm4/850 recvmsg(fd: 3<socket:[15058]>, msg: 0×ffffd788f1b0)                  = -1 EAGAIN (Resource temporarily unavailable)
1259.046 ( 4.100 ms): sudo/2502  ... [continued]: ppoll())                                           = 1
1260.262 (          ): xfwm4/850 ppoll(ufds: 0×aaaac93f37c0, nfds: 4)                             ...
1263.161 ( 0.011 ms): sudo/2502 rt_sigaction(sig: TTOU, act: 0×fffff9f0a718, oact: 0×fffff9f0a7b0, sigsetsize: 8) = 0
1263.184 ( 0.023 ms): sudo/2502 write(fd: 8</dev/tty>, buf: 0×aaaafbecc780, count: 48640)            = 5632
1263.235 ( 0.037 ms): sudo/2502 rt_sigaction(sig: TTOU, act: 0×fffff9f0a718, sigsetsize: 8)          = 0
1263.379 ( 3.388 ms): sudo/2502 ppoll(ufds: 0×aaaafbe9f2c0, nfds: 4, sigsetsize: 8)                  = 1
```

Figure 23 - Example of a raw system call log extracted with perf

- Data Preparation Module: In this step, the collected system call trace is processed. First, each system call is converted into a numeric equivalent. It is followed by a Feature Extraction step, where methods such as TF-IDF and N(2)-gram are applied in order to create a feature vector. Since the resulting vector is sparse, a Feature Selection method is applied, reducing the dimensionality of the input.
- ML Model: The resulting data is then sent as input to a Machine Learning (ML) model, more specifically to a MultiLayer Perceptron (MLP) model, for event classification. The model classification consists of a binary output, where 0 represents a benign behaviour and 1 represents malicious activity detected. Both outputs are presented with the respective confidence level.
- Event Handler: This subcomponent considers the output of the ML model, which is then forwarded to the RabbitMQ exchange pertinent and redirected to the relevant ARCADIAN-IoT components (e.g., CTI or Reputation System) or send to the Device Self-Protection (via inter-thread communication).

The developed code (written in Python3) and respective packages are available on the GitLab repository [50].

3.2.3 Design specification

This section considers the logical architecture view, where we describe the internal and external interfaces of the component, the sub use cases, sequence diagrams, deployment architecture model, interface description and resultant technical solution.

3.2.3.1 Logical architecture view

The Device Behaviour Monitoring oversees the security aspect on device level, this section presents the logical view of the internal and external interfaces which compose the component at hand. Figure 24 illustrates the logical architecture that follows the Device Behaviour Monitoring component, of both internal and external interfaces.

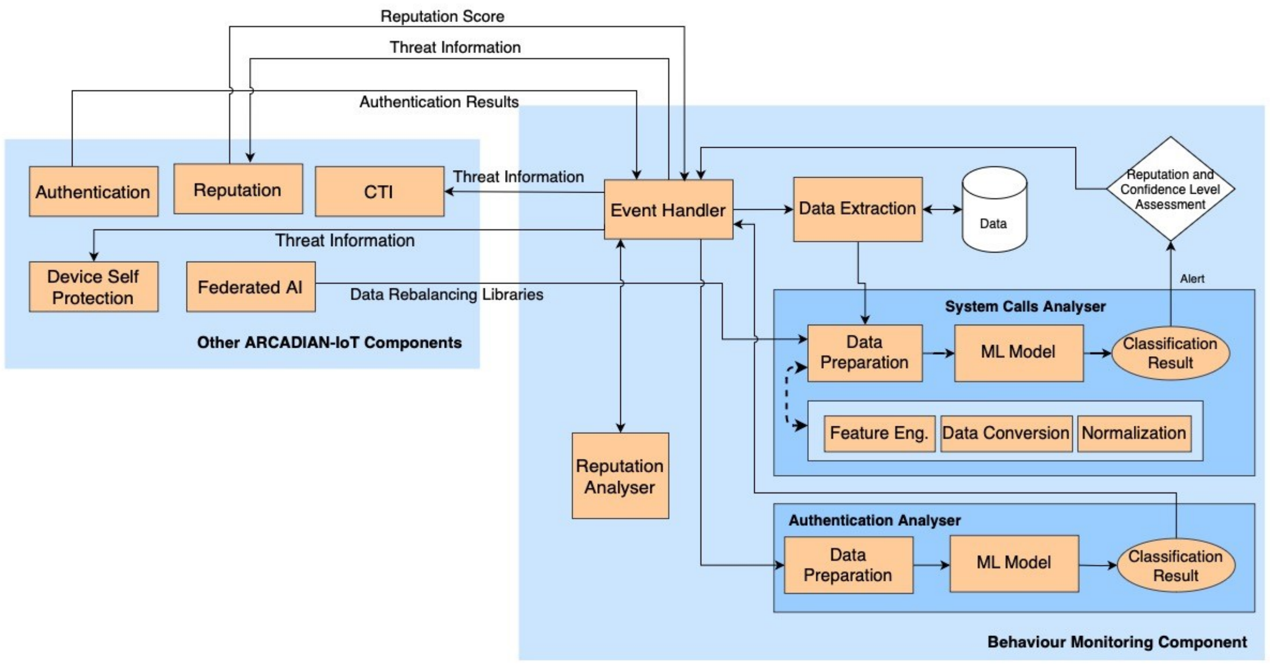


Figure 24 - Behaviour Monitoring component architecture

The internal workflow of the component is composed of the following sub-components:

- System Calls Analyser: Analyses sequences of system calls to detect anomalies, possibly linked to intrusion attacks.
- Authentication Analyser: Analyses sequences/time series of failed authentication events in search for possible brute force attacks.
- Reputation Analyser: Analyses sequences of reputation updates to detect abnormal, sudden changes in reputation scores.

Within the System Calls Analyser, we have the following elements:

- Data Extraction: Continuously extracts raw data directly from the device pertinent to user/device activity (e.g., system call traces) with the *perf (or strace)* Linux based tool for system call log activity (for Linux based devices).
- Data Preparation Module: The extracted raw data from the device and data from other ARCADIAN-IoT components (described in the external interfaces) is processed by this module and then sent to the detection ML model, which produces a classification of either normal or anomaly (from the input data).
- ML Module: The data from the Data Preparation Module is forwarded to the ML models that will indicate if it is an anomaly or not. If an anomaly is detected from the event traces, then the model will (when possible) classify what type of anomaly occurred.
- Event Handler/Alarm: The resulting classification of the anomaly detection module will be packaged along with other information, such as the level of confidence, process affected, and the service/program associated to the process. The report of the classification result to external interfaces is decided based on the reputation score of the device and the confidence level of the result (see above, in “Reputation Scores Usage in DBM“ section 3.2.2.2).

The Authentication Analyser works similarly (in general) to the System Calls Analyser, in that the purpose of the Data Preparation and the ML Model elements are mainly the same. However, information regarding authentication is received from the Event Handler which, in turn, received them from Authentication external interface. Additionally, contrary to the previous model, any anomalies in the Authentication are always sent.

The Reputation Analyser, as explained in the section 3.2.2.2, receives and analyses the history of reputation changes of the devices. Just like the Authentication Analyser, whenever it detects a sudden change, it reports it immediately.

3.2.3.2 Sequence diagrams

The sequence diagram in Figure 25 depicts the expected sequence of events in case of a security incident. The Device Behaviour Monitor component starts by examining the system call traces extracted from the host device. Whenever an anomaly is detected in the device's behaviour by the IDS model, an alert is generated by the Event Handler subcomponent (which is responsible for processing incoming and outcoming information) and forwarded to other ARCADIAN-IoT components, namely, the Device Self-Protection, Reputation System and CTI.

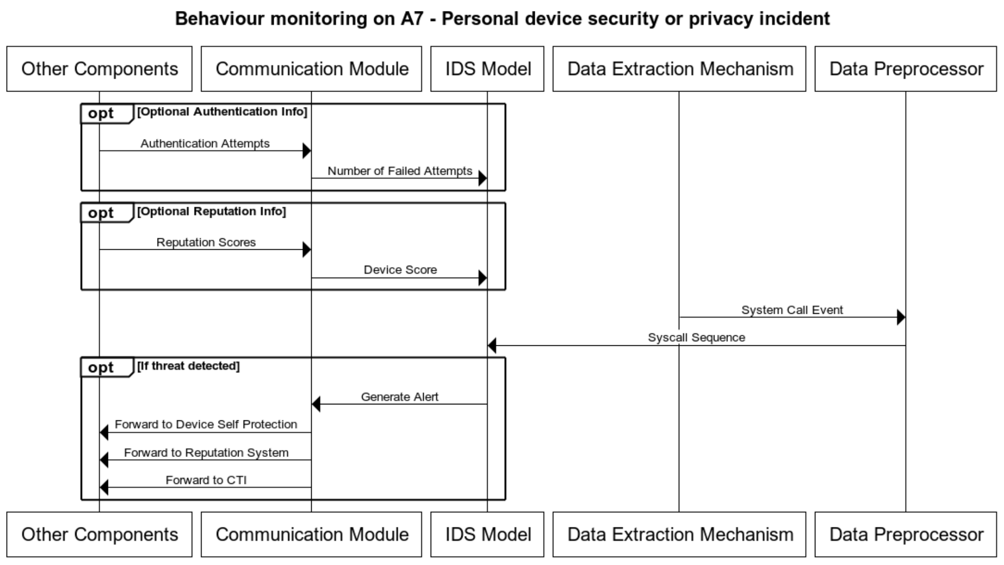

Behaviour monitoring on A7 - Personal device security or privacy incident
Other Components
Communication Module
IDS Model
Data Extraction Mechanism
Data Preprocessor
opt
[Optional Authentication Info]
Authentication Attempts
Number of Failed Attempts
opt
[Optional Reputation Info]
Reputation Scores
Device Score
System Call Event
Syscall Sequence
opt
[If threat detected]
Generate Alert
Forward to Device Self Protection
Forward to Reputation System
Forward to CTI

Figure 25 - Device Behaviour Monitoring Sequence Diagram

Furthermore, additional information related to the device at hand can be used in the process of classification of the events, such as the authentication events and device scores from the Reputation System.

3.2.3.3 Interface description

The external interfaces are the following:

- Alerts: Interface that will provide the results of the attack detection. It will use a protocol to send the alerts to a publication/subscription middleware via RabbitMQ or inter-thread communication. Other components such as the CTI, Device Self-Protection and Reputation system can then subscribe to such alerts and act accordingly.
- Reputation System: Interface designed to receive reputation updates (acting as a weighted input for the intrusion detection models) from the Reputation System and to send relevant alerts about device’s behaviour to that component.
- Federated AI: Access to Federated AI libraries for data rebalancing, which provides a mechanism to rebalance and fit non-Independent and Identically Distributed (non-IID) data to the framework by rebalancing the classes within a dataset, in the particular case of Behaviour Monitoring, balance the two classes (normal and anomalous behaviour).

- Authentication: Interface used to receive authentication results from the Authentication component (e.g., consecutive authentication attempts)

The following table details the external interfaces:

Partner	Definition	Channel	Exchange
IPN	Device Self Protection	RabbitMQ AMQP 0.9.1 and Inter-thread communication	Threat/intrusion detection warnings
UC	Reputation System	RabbitMQ AMQP 0.9.1	Threat/intrusion detection warnings
RISE	CTI	RabbitMQ AMQP 0.9.1	Threat/intrusion detection warnings
UC	Reputation System	RabbitMQ AMQP 0.9.1	Reputation updates
RISE	Federated AI	Direct Code Integration	Data Rebalancing Libraries
1GLOBAL	Multi-Factor Authentication (MFA)	RabbitMQ AMQP 0.9.1	Authentication Updates

Table 7 – Device Behaviour Monitoring External Interfaces

3.2.3.4 Technical solution

3.2.3.4.1 Deployment architecture view

The Device Behaviour Monitoring (DBM) component enables two main types of deployment. The first type of deployment (Figure 26), refers to a solution that is deployed directly on the device, drone or smartphone, which includes all subcomponents of the Device Behaviour Monitoring. The second type of deployment is deployed in the cloud or any other premise (Figure 27). The device-related information is sent via RabbitMQ to component, which processes the data.

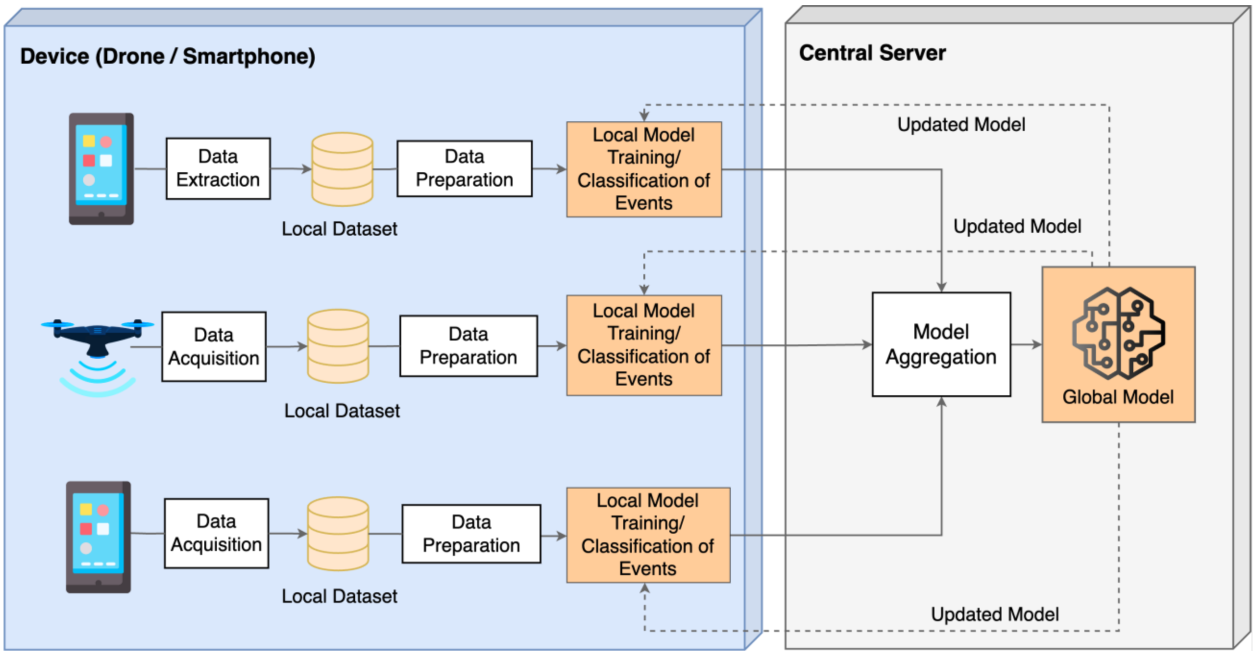


Figure 26 - On device deployment architecture view

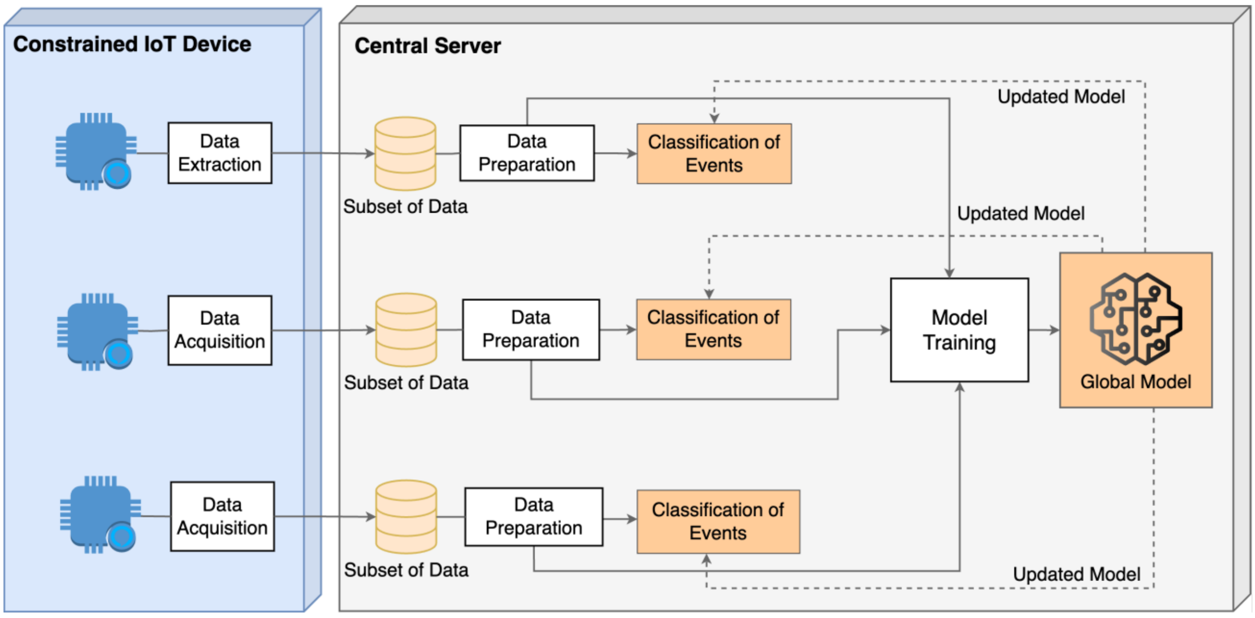


Figure 27 - Remote deployment architecture view

3.2.3.4.2 API specification

The Device Behaviour Monitoring component utilizes a RabbitMQ publish/subscribe model for exchanging information with other ARCADIAN-IoT framework components. It receives messages to analyse and sends messages with threat information, related to detected anomalies, to other components. The content of the messages sent, which detail the identified anomalies, vary depending on the nature of the anomaly, whether it's in system call sequences, an authentication anomaly, or a reputation anomaly. Table 8 specifies these details.

Message Format	Parameters	Type	Description	Format
JSON	timestamp	string	Current system time	Unix Timestamp (in seconds) (e.g., 1666620456)
	attack_start_date	string	Time when attack was first detected	Unix Timestamp (in seconds) (e.g., 1666620456)
	deviceId	string	arcadian_iot_ID of the device	Source:ID (e.g., ipn:3a45d35a-2494-407b-b8c9-adf39834a38f
	cause	string	Program/Service that was attacked	Service name (e.g., SSH, FTP, etc)
	processId	string	Process ID associated to the program	ID number of the process (e.g., 9346)
	Sender	string	Identification of sender component	<domain> <component> <location> <entity_id> <service>
	start_timestamp	string	Timestamp of the first failed authentication	Unix Timestamp (in seconds) (e.g., 1666620456)
	end_timestamp	string	Timestamp of the last failed authentication (in the same window)	Unix Timestamp (in seconds) (e.g., 1666620456)
	occurence_number	int	number of intrusions detected in the processId	A positive integer number.
	currentScore	float	Current reputation score of the device	A real number between, and including, 0 and 1
	previousScore	float	Current reputation score of the device	A real number between, and including, 0 and 1
	confidence_level	float	Model prediction confidence level	Float value from 0 and 1 (e.g., 0.55, 0.87, 0.98)

Table 8 - Device Behaviour Monitoring Output Specification

3.2.3.4.3 Security aspects

In addition to the inherent specificities of the component, is worth mentioning that the Device Behaviour Monitoring seeks to maximise the privacy of the devices targeted by the component. This is achieved by applying Federated Learning mechanisms when a model update is needed, this means that, device data is not shared with outside entities, as only the model parameters of the local device model are shared with a centralised server that performs model aggregation, keeping data secure and private.

Furthermore, the communication with other ARCADIAN-IoT components via RabbitMQ is authenticated – adding an additional security layer to the underlying operations of the component.

3.2.4 Evaluation and results

3.2.4.1 Model Performance

To evaluate the performance of the developed ML models and demonstrate the effectiveness of the proposed approach as it relates to the defined target KPIs, four performance metrics were selected. The performance of an IDS can be measured with the following metrics:

- **Accuracy** - The accuracy refers to the percentage of all those correctly predicted events in relation to all predictions.
- **True Positive Rate** (TPR) - TPR is the ratio of the number of correctly classified intrusions and the total number of true positives.
- **False Positive Rate** (FPR) - FPR is the ratio of the number of normal events predicted incorrectly as an intrusion and the total number of events.
- **F1-score** - Weighted average of the precision and recall.

Table 9 shows the performance of different machine learning algorithms on ADFA-LD dataset regarding performance indicators of accuracy, F1-Score, false positive rate, and true positive rate.

Algorithm	Accuracy	TPR	FPR	F1-Score
Logistic Regression	0.93	0.92	0.08	0.93
KNN	0.86	0.99	0.26	0.86
Decision Tree	0.97	0.97	0.04	0.97
Random Forest	0.99	0.98	0.02	0.98
SVM	0.96	0.94	0.06	0.96
Naive Bayes	0.73	0.82	0.36	0.72
Neural Network (MLP)	0.98	0.97	0.02	0.98

Table 9 - Model Training Results

Additionally, the application of the federated approach was also tested with the ADFA-LD dataset. In the evaluation of FL algorithm, two types of approaches were taken. In the first, the training phase of the MLP model was performed with data that was distributed equally among a pre-defined number of clients. In the second approach, the training phase is done with data that is randomly distributed among the clients. Table 10 shows the results obtained with the experimental work executed across both the scenarios.

Algorithm	Distribution	Accuracy	TPR	FPR	F1-Score
FedAvg	iid	0.96	0.97	0.03	0.96
FedAvg	non-iid	0.92	0.91	0.07	0.92
WeightedFedAvg	iid	0.96	0.97	0.03	0.96
WeightedFedAvg	non-iid	0.95	0.94	0.03	0.95

Table 10 - FL Model Training Results

Based on the values obtained, we were able to achieve similar results (i.e., slightly better) in comparison to the current state-of-art values for centralised training settings. Random Forest has showed to be 0.01% more accurate than the MLP.

Due to the hidden layers, we expected that the Neural Networks would be more accurate, however, NNs are generally more accurate when datasets are larger. In this case, ADFA-LD is large enough for a traditional ML algorithm to perform accurately, however it is not of an ideal size for the MLP.

Additionally, when the model is trained with the federated setting, the accuracy drops down to around 96% with Weighted Federated Averaging algorithm in comparison to the 98% when trained in a centralised setting.

3.2.4.2 Model Explainability

On an overall analysis of the results, it was possible to observe that there are system calls that only appear, for example, in the explanations of FN cases, and not in the explanations of TP cases, and vice versa. The same was observed for the FP and TN cases. The results and consequent system calls are represented in Figure 28. It is important to note that the represented results match with LIME and SHAP.

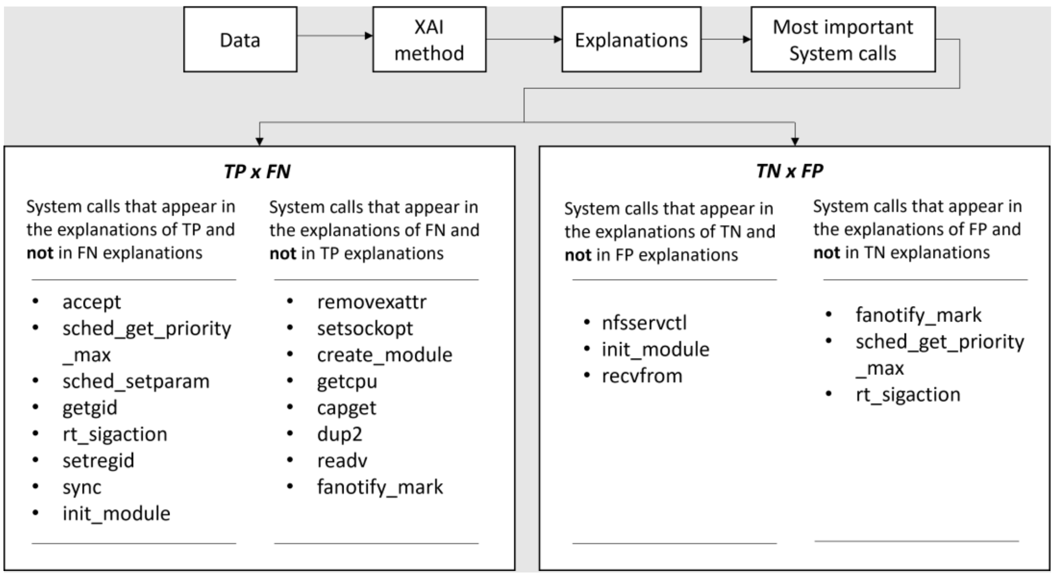

Data
XAI method
Explanations
Most important System calls
TP x FN
System calls that appear in the explanations of TP and not in FN explanations
accept
sched_get_priority_max
sched_setparam
getgid
rt_sigaction
setregid
sync
init_module
System calls that appear in the explanations of FN and not in TP explanations
removexattr
setsockopt
create_module
getcpu
capget
dup2
readv
fanotify_mark
TN x FP
System calls that appear in the explanations of TN and not in FP explanations
nfsservctl
init_module
recvfrom
System calls that appear in the explanations of FP and not in TN explanations
fanotify_mark
sched_get_priority_max
rt_sigaction

Figure 28 - System calls that only appear in the explanations of particular cases.

To evaluate and validate the explanations provided, we have induced perturbations based on the most important system calls. A system call that only appears, for instance, in the explanations of FN, was replaced with a system call that only appears in the explanations of the opposite case (i.e., TP). The same was performed for the TN and FP cases. Some of the most interesting results are represented in Figure 28 and Figure 29.

With the explanations provided by the XAI methods, and the perturbations that can be made according to the explanations, the interpretability of the model's results increase, and the cybersecurity personnel in charge of analysing the results might get insights into the model's behaviour, and what system calls might be having more impact for different scenarios. With the results, it can be concluded that there are system calls that might be causing a negative impact on the model's classifications. With the knowledge acquired about the model's behaviour in the presence of such system calls in the input sequences, the cybersecurity personnel in charge of the analysis of the results might have a different perspective on what might be wrong.

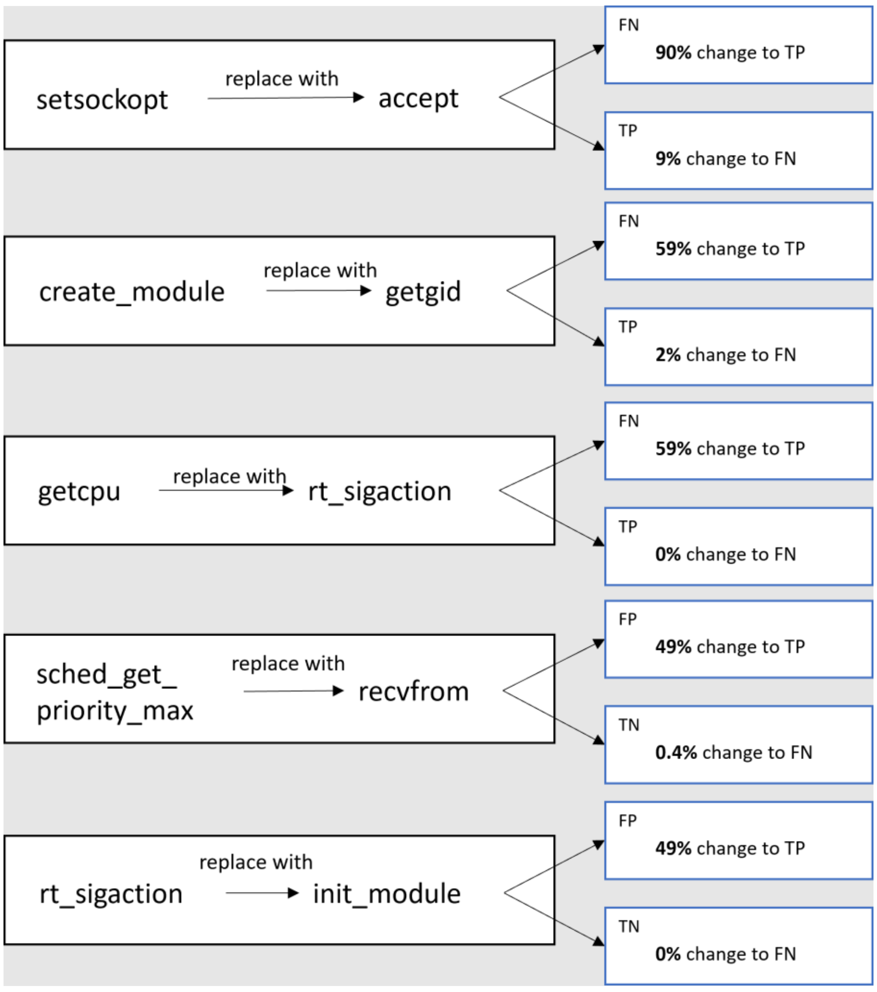


Figure 29 - System calls that only appear in the explanations of particular cases.

3.2.4.3 Computational Performance

To evaluate the performance overhead when the intrusion system is deployed on an emulated resource restricted system, we first evaluated the average CPU usage over 30 minutes. During this period of time, the emulated system is not running any high demanding processes, which gave us a baseline to compare to when the IDS program is deployed.

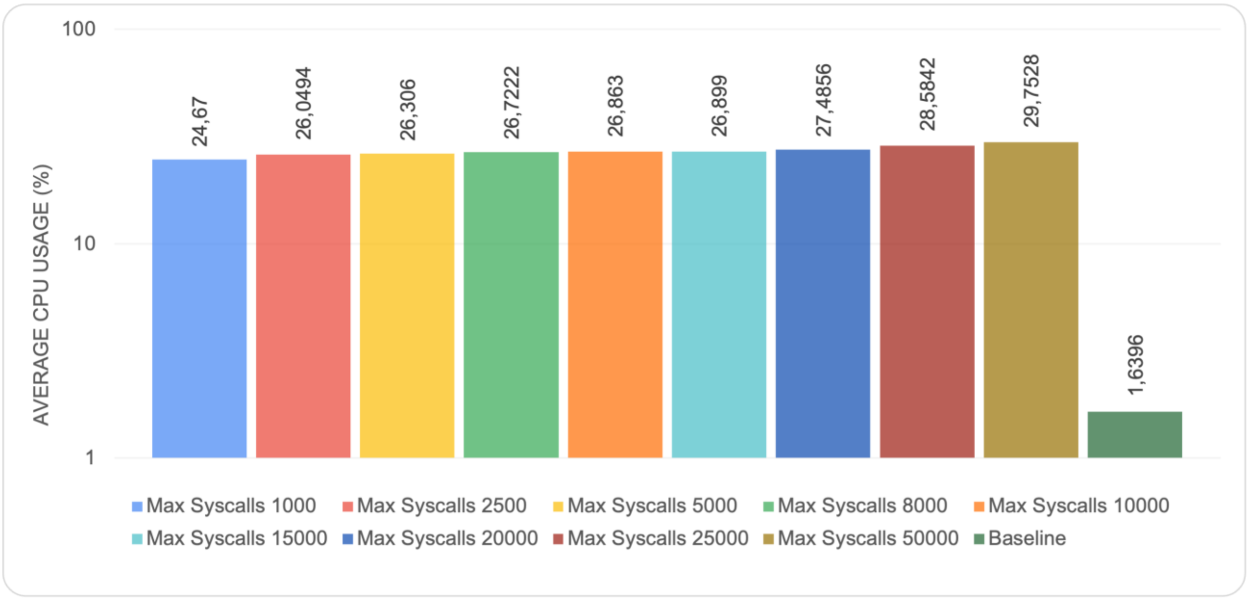


Figure 30 - Device Behaviour Monitoring CPU Usage

From the results we can observe that the more syscalls are continuously processed, the more CPU it requires. This means that if we intend to make the system less taxing on the device, we can always tune the amount of analysed system calls as a batch more limited. Nevertheless, we see that by increasing 10 times the number of system calls, we only see a 5% increase in CPU usage. In systems where this amount of CPU usage is not acceptable due to energy constraints, it is possible to disable the system call anomaly detector and still rely on the capabilities of the anomaly detector based on reputation and authentication inputs.

3.2.5 Future work

Since the component was finalised and integrated into various use cases within the three domains of the project, the subsequent work associated to this component, beyond the scope of the project, may be to focus on expanding integration capabilities (e.g., different operating systems and device types), adjusting the functionalities (e.g., supporting multi-class models) or supporting additional inputs for analysis.

3.3 Cyber Threat Intelligence

The design and development of the Cyber Threat Intelligence component is part of Task 3.4 (Cyber Threat Intelligence for IoT systems) in WP3.

3.3.1 Overview

3.3.1.1 Description

ARCADIAN-IoT intends to provide an instrument to gather, produce, elaborate, and share information regarding cyber threats and attacks in the IoT domain where end devices might be critically affected. The threat information is generally presented as Indicator of Compromise (IoC), and it can be shared between various partners to detect similar attacks in other organizations, or directly used to detect and analyse new security incidents.

RISE has built up an IoT-specific Cyber Threat Intelligence (CTI) system based on Malware Information Sharing Platform (MISP),[1] an open-source threat sharing platform, to orchestrate the process of (i) information parsing, formatting, and sharing, (ii) consuming and generating IoCs, and (iii) fetching feeds to CTI. Although MISP already provides a bunch of useful functionalities to administrate cyber threat data, there are still some critical features missing, such as, IoC quality control, and automatization which would be useful in IoT environments. Understanding the quality/reliability of IoCs, having an appropriate level of contextualization in different organizations are challenging tasks. Thus, essential robust mechanisms are needed. Additionally, exploiting the Federated AI Component (Section 2.2), ML-based models enable the sharing of IoCs among multiple instances in a privacy-preserving manner. CTI platform will enable the direct use of IoCs for anomaly detection, intrusion detection and prevention, and designing of novel protection mechanisms.

An IoT-specific CTI is responsible for collecting, processing, and sharing IoCs regarding to cyber threat within IoT network. Different features have been identified as needed for the essential functionalities to meet the use case requirements in ARCADIAN-IoT.

3.3.1.2 Requirements

A recall of the high-level requirements that have been previously defined and provided in deliverable D2.4 [2] is given below.

Requirement 3.3.1 – Threat data collection: The CTI should be able to collect threat data from various sources, local and internal sources (wide variety of various sources).

Requirement 3.3.2 – Private information: The CTI may need to share information about compromises; Local intelligence in application to not disclose sensitive/confidential information belonging to users or company.

Requirement 3.3.3 – Indicators of Compromise: The CTI should support Indicator of Compromise (IoC) generation and sharing by any participating IoT or edge device.

3.3.1.3 Objectives and KPIs

The aim of the ARCADIAN-IoT project is to provide a MISP-based CTI platform focused on IoT-specific threat intelligence. With the aim of fulfilling the project's objectives, the CTI will be evaluated against the following KPIs:

KPI scope	
Enable the aggregation and processing of IoCs generated by internal or external sources	
Measurable Indicator	
Number of the types of threat managed Number of different IoC formats supported	
Target value	***Achieved value***
# types of threat > 5 # IoC formats > 3	# types of threat = 12 # IoC formats = 4

KPI scope	
Provide ML models for automated IoC clustering	
Measurable Indicator	
Purity, Silhouette Coefficient, Normalized Mutual information	
Target value	***Achieved value***
Purity > 70% Silhouette Coefficient > 0.7 [-1, 1] Normalized Mutual information > 0.7 [0, 1]	Putity ~= 89.86%

	Silhouette Coefficient ~= -0.1331[16] Normalized Mutual information ~= 0.7864

KPI scope	
Provide ML models for automated IoC ranking	
Measurable Indicator	
NDCG, MAP@10, Precison@10	
Target value	***Achieved value***
NDCG > 80% MAP@10 > 80% Precison@10 > 80%	NDCG = 98% MAP@10 = 100% Precision@10 = 100%

KPI scope	
Provide ML models for automated Event/Galaxy classification	
Measurable Indicator	
Accuracy, F1-score	
Target value	***Achieved value***
Accuracy > 85% F1 > 0.85	Evaluated with 3 datasets: Average accuracy: 89.44% Average F1 score: 0.8807

3.3.2 Technology research

3.3.2.1 Background

Despite technologies, tools, and best practices for threat data sharing are quite consolidated, automated processing of CTI platforms is an area where research is still advancing, especially in sectors including critical services, such as healthcare, banking, energy, and transportation. Along with the growth of IoT, several ongoing projects and standardization activities are working on new lightweight IoT security protocols, secure connectivity of IoT with cloud backend, distributed trust in IT, etc. However, CTI focused on IoT is a relatively immature discipline; in fact, most current CTI platforms focus on standard internet hosts.

MISP

ARCADIAN-IoT CTI system leverages on MISP platform. In MISP, the structure of shared information is well-defined. The terminology in MISP can briefly be categorized into two classes: data layer and context layer. The former includes all the terms related to how the information is defined in MISP; the latter includes the terms referred to the relationship between different clusters of information. In details, the data layer contains:

- Event: The encapsulation for contextually linked information represented as attribute and object
- Attribute: The individual data points, which can be indicators or supporting data

[16] The silhouette coefficient measures the distance between clusters at the end of the clusterization. The low silhouette coefficient in this case is due to the type of dataset utilized in the experiments, which are with high dimensionality and causes a high number of clusters so that the distance between them is too close.

- Object: The custom template for attributes
- Object reference: The relationship between other building blocks
- Sighting: The time specific occurrences of a given data-point detected.

And the context layer includes:

- Tags: The labels attached to events/attributes from taxonomies
- Galaxy-clusters: The knowledge base item used to label events/attributes and come from Galaxy
- Cluster relationship: The relationship between Galaxy clusters.

The IoC, which is a piece of information that helps Intrusion Prevention/Detection Systems and their administrators to detect suspicious or malicious cyber activities, can be generated based on this data structure. IoC usually involves different attributes related to the object of an event, and it can be represented as network indicator (e.g., IP address), system indicator (e.g., string in memory), or bank account detail. However, there are no events, attributes, and objects yet specifically designed for IoT contexts. By analysing the three project domains and their specific requirements, we are able to define a uniform set of IoT-specific events, attributes, objects to include in IoT-specific IoCs.

STIX and TAXII

The Mitre Corporation's efforts on structured threat information expression (STIX) and trusted, automated exchange of indicator information (TAXII) are prominent Cyber Threat Intelligence (CTI) frameworks and platforms that are being used today [i]. In general, CTIs represent Indicator of Compromise (IoC) for formalizing and/or representing threat actors and attack-vectors. STIX is the most prominent and preferred CTI framework being used by hundreds of CSOs worldwide nowadays, for protecting enterprises, institutes, national assets, etc. against Advanced Persistent Threats (APTs) by sharing attack specific IoC amongst peers. The main question we are seeking answers to in this work is this:

"How can we adapt this technology to the IoT with the requirement of less energy consumption?"

Each STIX object is on average hundreds of bytes on the disk, which might be considered negligible for traditional networks. However, in the case of an IoT network, if several IoC 's need to be broadcast per day, then might significantly increase the overall power consumption of the network, as it is measured at milli-Watts range. Hence battery level of the end-devices is the most important constraint in IoT networks, we need to find a way to decrease the STIX message size to be transmitted over.

STIX v2 is in JavaScript Object Notation (JSON) format, compared to the earlier version (STIX v1) which was in eXtensible Markup Language (XML) format. As such, all CTI information related to IoCs are represented in JSON file format consisting of STIX object(s) fields.

Trusted Automated Exchange of Intelligence Information (TAXII) is a platform devised for automated CTI sharing based on STIX message formatting. The features include:

- Negotiations and authentications is based on HTTP protocol
- Existing protocols are exploited by TAXII whenever possible
- DNS service records are used to discover TAXII servers within a network
- All TAXII exchange types use JSON (UTF-8 encoded) as serialization format
- All TAXII communications are transported via HTTPS

3.3.2.2 Research findings and achievements

Preliminary adaptations of MISP platform for IoT

Through the MISP platform, a user can manually insert, edit, delete, and search for events or objects. Users also can define how to share the information by selecting one of multiple approaches provided by MISP, e.g., a user may define the profile of the organizations with which the information will be shared and the sharing time frame. The information included in the shared IoCs is constrained by the IoT-specific context and fulfils the requirement of threat data required by other components in the framework (MISP4IoT). MISP platform has been extended with automatic control functionalities (e.g., automatic insert, edit, delete, search). For example, we provide an API for automatic creation of threat events when an IDS event is raised by the flow/Behaviour Monitoring component.

The new functionalities have been implemented in Python by utilizing PyMISP library. PyMISP provides essential REST APIs to access to MISP platform. It allows us to develop automatic-control functions of MISP, and these functions are essential for future exploitation of the others subcomponents in CTI platform. Through these functions, the threat data can be automatically added, updated, deleted. The necessary information includes CTI platform's IP address, the content of attributes to the defined object, and the authorized key, and some additional parameters (e.g., threat level, distribution, IDS flag, etc.).

TinySTIX and TinyTAXII for CTI in IoT

Besides the implementation, we have investigated the most suitable data formats and communication protocols for IoT domains. Starting from the well-known language and serialization format STIX (Structured Threat Information Expression), which is an open-source format and integrates well with MISP for exchanging CTI data, we implemented a simplified and lightweight version of it to exchange threat data within IoT domains. This format helps us easily contributing to and consuming from the CTI platform as all dimensions of suspicion, compromise, and attribution can be emphasized with descriptive relations along with the object identifiers. It is presented as JSON, which is a machine-readable format, and it can be visualized in a graphical representation. On the other hand, we exploit the TAXII (Trusted Automated Sharing of Intelligence Information) protocol, which was designed to support STIX, in order to tailor the methods used for sharing cyber threat information in the three ARCADIAN-IoT domains.

RISE has also investigated approaches of reducing STIX message size so that to represent it in a compact and IoT-friendly form. As such, a new lightweight STIX format, called TinySTIX, was defined. First, some specific filtering of the required and optional fields of the STIX objects, represented as binary listings, has been performed. For example, the "*type*" field of the STIX objects can be applied with Binary Listing methodology to reduce the size. (e.g., after applying the binary listing methodology to the type field of the STIX objects with the TinySTIX, we now have a size reduction rate of 1.85%). There are more fields are available to be applied with binary listing method, such as "*spec version*": Version of the STIX framework, "*indicator types*": Mentions the type of the IoC, "*relation type*": Mentions the relation of the IoC, "*pattern type*": Mentions the pattern type of the IoC, "*pattern version*": Mentions the pattern version of the IoC, "*malware types*": Mentions the malware type associated with that specific IoC.

On the other hand, some specific selection of the required and optional fields of the STIX objects can be represented in an encoded or shortened format. As an example, we argue that the IoT devices do not generate precise timing and their representing IoCs do not need to have milli-seconds precision and can be shown only in seconds. We observed that, even such a small trick saved us 24 bytes space in the overall message size which is equivalent to 2.2% size reduction. A new lightweight TAXII format, named TinyTAXII, is also proposed which uses CBOR serialization rather than JSON serialization, and help reducing the packet sizes around 25% on average. The Concise Binary Object Representation (CBOR) is a data format whose design goals include the possibility of extremely small code size, fairly small message size, and extensibility without the need for version negotiation.

ML-assisted processing for CTI management

RISE has investigated the use of ML to solve three key challenges in CTI processing and management for IoT: prioritizing and ranking received Indicators of Compromise (IoCs) based on their severity, likelihood of exploitation, and potential impact; classifying the IoCs based on the type of attacks or threats; and aggregating IoCs with similar characteristics into clusters.

With this objective, RISE has extended the popular open-source CTI platform MISP (Malware Information Sharing Platform) to include ML functions. Our extension integrates three types of ML models for IoC ranking, classification, and clustering.

RISE has defined and developed three different ML-based modules to be used for smart IoCs processing and analysis: (i) threat ranking, (ii) event classification, and (iii) IoC clustering.

More details about each component of the CTI system are provided in the logical architecture view in Section 3.3.3.1.

3.3.2.3 Produced resources

A research paper on the CTI system and the obtained results is currently being peer-reviewed at the time of writing this deliverable. The source code of the system will be made publicly available in the ARCADIAN-IoT public repository when the paper is accepted.

3.3.3 Design specification

3.3.3.1 Logical architecture view

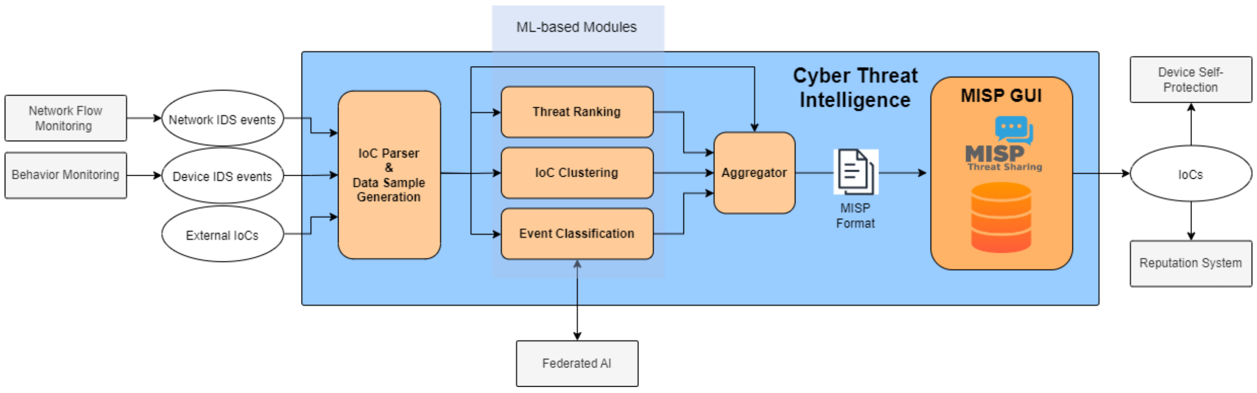


Figure 31 - The architecture of CTI

Figure 31 shows the architecture of CTI which consists of: an IoC parser, a threat ranking module, a threat classification module, an IoC clustering module, a data aggregator, the MISP engine, and a database specifically dedicated to storing IoCs in MISP format. Each subcomponent is in charge of different tasks.

- The **IoC parser** is responsible for processing and extracting relevant information from the IoCs received from different sources (internal and external).
- **Data Sample Generation**: after processing the IoCs, the parser generates a data sample for each IoC instance containing the extracted information. This data sample serves as input for the three ML-based modules. This step involves feature extraction and data pre-processing which are also required in the ML model training.
- The **threat ranking module** is trained to prioritize and rank the received IoCs based on their severity, likelihood of exploitation, and potential impact. If the rank value is missing from the data sample, the threat ranking module makes a decision on this specific feature. The module takes as input a data sample generated by the sample generator. Depending on the type of input, whether resulted from STIX or MISP formats, the threat ranking module selects a specific pre-trained unbiased Learning-to-Rank (LTR) model tailored to the respective data structure. In order to learn the threat level order of IoCs, we exploit

ranking algorithms.

- The **event classification module** is designed to classify the type of threat, such as malware, phishing, or DDoS attacks. Its purpose is to assist with data samples that lack event names, which often correspond to unidentified malicious activities or instances with unknown events. This module has the capability to assign pre-defined names to the IoCs, thereby providing valuable insights and identification for previously unclassified threats. The module takes as input a data sample generated by the sample generator. Depending on the type of input, whether resulted from STIX or MISP formats, the event classification module selects a specific pre-trained classification model, among those pre-configured, tailored to the respective data structure. In the centralized setup, we can execute four ML-based algorithms: DNN, CNN, XGboost, and RF. Instead, the setting with federated learning implements DNN and CNN.
- The **IoC clustering module** is responsible for grouping similar IoCs together based on common attributes or behavior patterns. If the cluster value is missing from the data sample, the IoC clustering module makes a decision on this specific feature. The module takes as input a data sample generated by the sample generator. Depending on the type of input, whether resulting from STIX or MISP formats, the IoC clustering module selects a specific pre-trained unsupervised clustering model among those pre-configured, tailored to the respective data structure. The module assigns the input data sample to the appropriate cluster based on its similarity to the existing clusters obtained during the training phase. The inference first outputs a tag with the cluster ID assignment to the input IoC for the further use when the IoC is being shared. The IoC is not shared with the users/organizations that contributed. The inference output also includes a confidence score indicating the level of certainty in the cluster assignment. This score represents the similarity between the input IoC and the centroid of the assigned cluster.
- **Aggregation and MISP Conversion**: the outputs from the three ML modules, along with the original input data sample, are aggregated together by the data aggregator. The aggregated data is then converted into MISP format.
- **MISP Database**: the updated and converted data sample is stored in the MISP database, which is dedicated to storing IoCs in MISP format.
- The **MISP engine** handles the sharing of IoCs in MISP format according to the approaches provided by MISP. Since the system is designed as an extension of the MISP platform, it inherits MISP's concept of a "sharing group." MISP engine also provides the graphic user interface (GUI) that displays threat intelligence.

3.3.3.2 Sequence diagrams

Figure 32 describes the sequence diagram of the CTI system which includes the functions for IoC parser and data generation, aggregator, threat ranking, event classification, IoC clustering, and MISP database. When the CTI system receives an alert from one of the ARCADIAN-IoT components or from an external source, the parser will extract its features and generate the data sample. If there is no threat level associated with the data sample, a threat level is automatically assigned by the ML-based threat ranker before being transferred to Aggregator. If the event is unlabelled, i.e. no event type is associated with the data sample, a label is automatically assigned by event classification module before being transferred to the Aggregator. A new tag, which is a cluster ID, is added to the data sample. Aggregator aggregates all the newly assigned features to generate IoC. The IoCs processed by the MISP core are then pushed towards the output API which delivers IoCs to the subscribers according to their tags/galaxies.

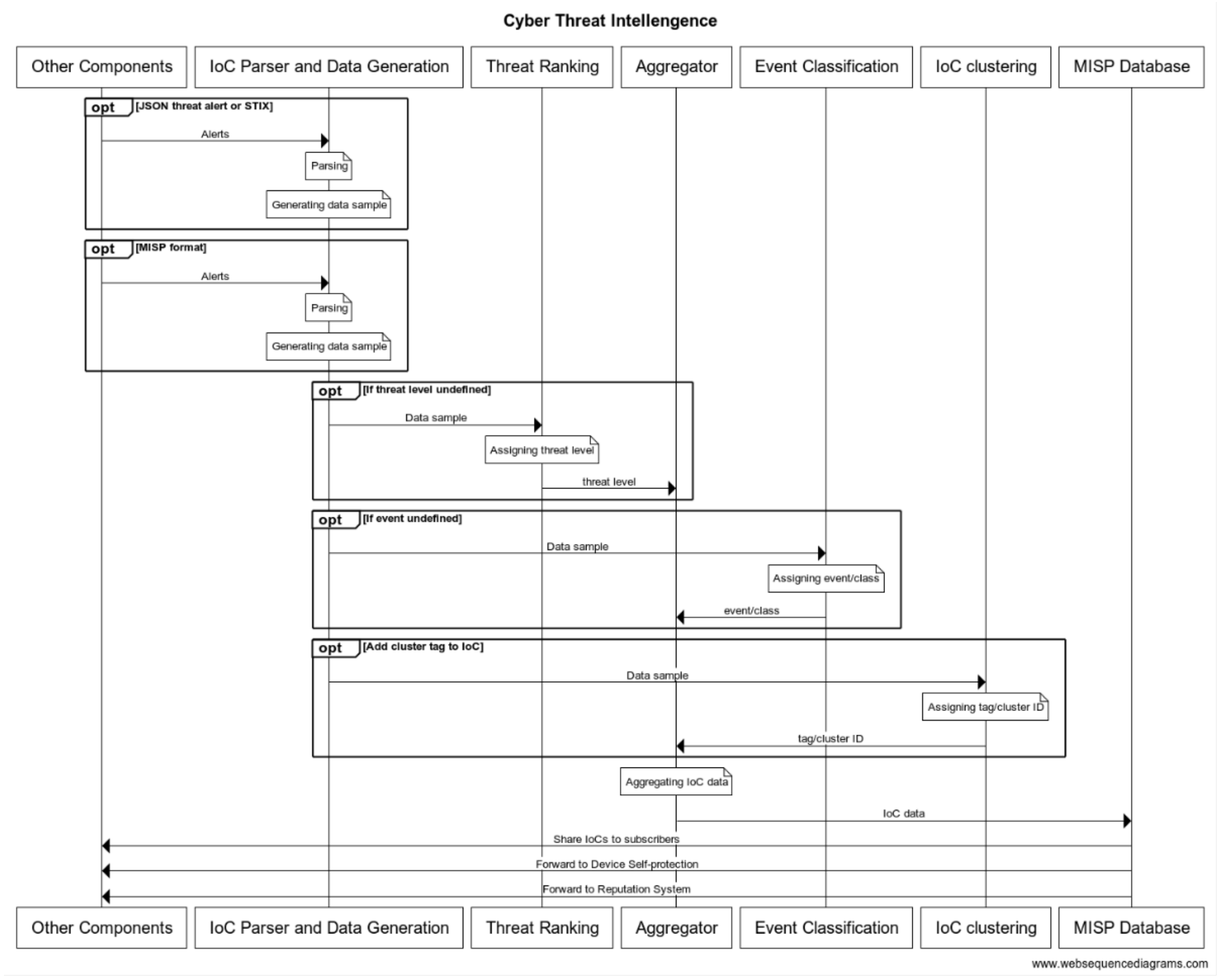


Figure 32 - Sequence diagram

3.3.3.3 Interface description

The external interfaces are the following:

- Indicator of compromise: Interface that will share the IoC resulted from the CTI processing. CTI uses a protocol to send the alerts to a publication/subscription middleware via RabbitMQ. Other components such as the IoT Device Self-protection and Reputation system can then subscribe to such IoCs and act accordingly.
- Behaviour Monitoring and Network Flow Monitoring send alerts as results of the attack detection. It uses a protocol to send the alerts to a publication/subscription middleware via RabbitMQ. CTI system subscribes to such alerts and processes the alerts for further analysis and aggregation.
- Federated AI: CTI have access to Federated AI libraries for data rebalancing; the component is used to rebalance the datasets in the federated communication.

The following table details the external interfaces:

Partner	Definition	Channel	Exchange
UWS	Network Flow Monitoring	RabbitMQ AMQP 0.9.1	Threat/intrusion detection warnings
IPN	Behaviour Monitoring	RabbitMQ AMQP 0.9.1	Threat/intrusion detection warnings

IPN	Reputation System	RabbitMQ AMQP 0.9.1	Indicators of Compromise
IPN	IoT Device Self-protection	RabbitMQ AMQP 0.9.1	Indicators of Compromise
RISE	Federated AI	Direct Code Integration	Data rebalancer libraries

Table 11 - The external interfaces of the CTI component

3.3.3.4 Technical solution

3.3.3.4.1 API specification

CTI system is using a message bus publication/subscription channel to interact with the rest of the components in the ARCADIAN-IoT framework. This component uses two different exchanges: the first is for Reputation System, and the second is for Device Self-protection. These two components are receiving IoC from CTI component. As a second interface, the CTI component subscribes to the Network Flow Monitoring and Device Behaviour Monitoring to consume threat alerts to process in CTI system. The format of IoC is detailed in the following Table. The optional fields will be provided as other components require.

Output IoC Format (JSON)			
Name	**Description**	**Type**	**Requirement**
Timestamp	Time when event was first created	String	mandatory
Event_class	The class information of the event	String	mandatory
Threat_level	The threat level of the event	Integer	mandatory
Attribute_type	The type of the information in the IoC	String	mandatory
Attribute_value	The value of the information/attribute	String	mandatory
Attribute_comment	The comment of the attribute	String	optional
Source_Org	The contributer of the event	String	optional
aiotID	ARCADIAN ID of a device (if available)	String	optional
srcIP	Source IP as provided by network flow monitor (if available)	String	optional
dstIP	Destination IP as provided by network flow monitor (if available)	String	optional

3.3.4 Evaluation and results

In this section, we evaluate and compare the performance of the selected ML models for each ML-based module. We first describe the processes of ground truth labelling, dataset preparation and model tuning that were implemented for evaluation. Then, we present the numerical results obtained from our evaluation.

We tested the three ML-based modules of the CTI system on three publicly available CTI datasets: (i) Mitre ATT&CK,[17] (ii) CIRCL,[18] and (iii) Botvrij.[19] While the Mitre ATT&CK dataset comprises IoCs in STIX format, CIRCL, and Botvrij comprise IoCs in MISP format.

The Mitre dataset does not provide threat levels for its entries. Therefore, we extract relevant information from the "target references" field defined in the relationship object of the STIX format to assign both threat levels and event categories. The "target references" consistently points to the attack pattern, which is a domain object containing Mitre-defined fields such as "kill_chain_phases," "x_mitre_data_sources," and "x_mitre_platforms," among others. We extract numerical information from these fields. For instance, we assume that an attack pattern used on many platforms is more critical than others. Similarly, we consider an attack pattern with information provided by multiple data sources to have been discovered multiple times, making it more significant. The field named "kill_chain_phases" refers to the tactic information, such as credential access, reconnaissance, and so on. Each tactic has a various level of importance. By extracting and analyzing this information, our goal is to assign appropriate threat levels to the data entries, enhancing the evaluation of threat ranking in the Mitre dataset. In order to do this, we have assigned the weight for each field and determined the importance of tactics in the assignment of the threat level. For the evaluation of event classification, we use tactic information as the event categories.

CIRCL and Botvrij datasets are both in MISP format and include the ground truth for the threat level. However, they do not provide event categories that can be used as ground truth for the evaluation of event classification. Therefore, we manually labelled the events by analyzing the information provided in the "attribute comment," "event name," and any other informative fields of the data instance.

The three datasets used in this evaluation have been divided into training and testing subsets. A standard ratio of 70-30 train-test split strategy is employed, where a certain percentage of each dataset is randomly selected for training the ML models, while the remaining data is reserved for testing and evaluation.

In the development of the ML-based models, we followed the default settings as defined by the applied APIs for most of the algorithms. These default settings serve as a starting point and provide a general configuration suitable for various tasks. However, when it came to neural networks included in the models, we encountered the need for parametrization and fine-tuning to optimize their performance for specific problems and models.

In order to evaluate different ML-based tasks, different metrics are exploited to assess the model performance.

We measure the effectiveness of an unbiased LTR model in terms of nDCG@10, which can be formulated as:

$\$nDCG@10 = \frac{\{DCG@10\}}{IDCG@10}$, where DCG@10 is the measure evaluating the quality of a ranking of 10 items, and IDCG@10 is the ideal DCG@10, which is the DCG (Discounted Cumulative Gain) value of the best possible ranking of 10 items.

For the event classification problem, we evaluate the accuracy; since the data is slightly imbalanced to several dominated classes, we additionally adopt Marco F1 score (FS-marco) and geometric mean (G-mean).

Finally, the performance evaluation of the IoC clustering module is based on three metrics: homogeneity (HO), mutual information (MI), and rand index (RI). HO assesses whether a cluster exclusively contains instances from a single class, which is the purity of the cluster. MI measures

[17] https://attack.mitre.org/
[18] https://www.circl.lu/doc/misp/feed-osint/
[19] https://www.botvrij.eu/data/feed-osint/

the similarity between two distinct sets, namely the labels and clusters. A higher MI value signifies a stronger correlation between the labels and clusters, indicating a more certain distribution. The rand index quantifies the proximity between two clusters.

First, the performance of nDCG@10 and accuracy on test dataset for Botvrij, CIRCL, and Mitre datasets are evaluated. The results show DCG@10 values higher than 98% which indicates that we can expect the unbiased LTR algorithms to produce good ranking values. On the other hand, the accuracy results show that the unbiased LTR models can overall achieve over 82% accuracy among the three datasets. It shows the module can return the promising threat level.

Second, we provide an overview of the overall performance of the classification models across three metrics on three datasets. A notable observation is that the ML models obtain relatively lower performance on the Mitre dataset compared to the other datasets. The highest accuracy achieved for the Mitre dataset is only 85% by the RF model, and FS-Marco also displays a similar value. It is due to the nature of the three datasets are different. In contrast, the rest of the datasets achieve higher accuracy levels, surpassing 90% for both accuracy and FS-Marco. This is likely due to the datasets containing less intricate and more clearly delineated threat event classes, enabling the ML models to perform better in classification tasks.

Last, clustering models perform reasonably well, especially on the CIRCL and Botvrij datasets. The obtained HO value indicates that about 85% of the IoCs within a single cluster are related to the same labelled event, demonstrating the clustering model's effectiveness. MI shows that the clustered IoCs exhibit more than 75% correlation with the provided labels, further validating the performance of the clustering approach on these datasets.

3.3.5 Future work

Future work beyond the timeframe of WP3 could be dedicated to enhancing the overall CTI system. The primary objective will involve adjusting the CTI system to accommodate a wider range of data formats for IoC or threat alerts, as well as expanding the sources of data. Additionally, efforts may be directed towards exploring techniques for extracting threat intelligence from social media, utilizing pre-trained LLM (Large Language Model), and conducting further analysis on this data to generate valuable IoCs.

3.4 Network Self-protection

The design and development of the Network Self-Protection component is part of Task 3.5 (Self-healing and self-protection for IoT systems) within WP3 which addresses the horizontal plane of the ARCADIAN-IoT framework.

3.4.1 Overview

3.4.1.1 Description

This component is responsible for providing the self-protection capabilities required by the ARCADIAN-IoT framework in the wired segments of the network (e.g., Core, Edge and Transport). This component has to enforce a set of protection policies into the software data plane with the aim of safeguarding and protecting the network infrastructure, the IoT devices, and the services against volumetric DDoS attacks. For this purpose, this component is hooked into the data plane providing an enhanced programmable enabler beyond the current state of the art able to enforce security policies by means of protection rules. The implementation is based on the well-known virtual switch OpenVSWitch (OVS) on which the UWS team is carrying out significant extensions and upgrades to support an enhanced programmability of the data plane to meet the expected requirements of the project. Thus, the component has been designed and implemented to cope with different data paths and network protocols related to IoT networks, overlay networks, or any other type of networks, scenarios and use cases involved in the ARCADIAN-IoT project.

This component, in cooperation with the Network Flow Monitoring (Subsection 3.1) and the Network Self-healing (Subsection 3.6) components, performs an autonomous cognitive loop able to detect and mitigate known cyber-attacks (volumetric DDoS attacks). First, the Network Flow Monitoring detects the attack and raises an alert. This alert is received by the Network Self-healing that determines what action or set of actions needs to be executed to stop the attack (which rule(s) to enforce in the data plane and where in the network topology they will be inserted). Finally, the Network Self-protection is responsible for executing such self-healing rules in the data plane and thus, stopping the attack and restoring the network traffic back to pre-attack performance The Network Self-Protection architecture consists of three main sub-components:

- Protection Decider (PD)
- Data Security Controller (DSC)
- Protection Control Agent (PCA)

The functionality of these architectural sub-components is discussed in detail in section 3.4.3.1. describing the logical architectural view of the component.

3.4.1.2 Requirements

A recall of the high-level requirement that has been previously defined and provided in deliverable D2.4 is given below.

Requirement 8.2.1 – Mitigate Attack against IoT Overlay Networks: The Network Self-protection component will be deployed in the Edge and Core segments of the ARCADIAN-IoT infrastructure. In these segments, network traffic sent by IoT devices and sensors may be processed and encapsulated in overlay networks to guarantee user mobility and isolation between different tenants or users sharing the same physical infrastructure. Therefore, this component must be able to deal with overlay traffic and to identify the traffic from any IoT device regardless the number and types of encapsulation headers.

3.4.1.3 Objectives and KPIs

As stated above, the main aim of the Network Self-Protection component is to provide protection capabilities that will allow to fulfil the specific requirements of the different use cases and scenarios addressed in ARCADIAN-IoT. To assess the suitability of the component, the following KPIs will be used to validate its performance:

KPI scope	
Novel traffic classifier able to deal with data paths in 4G/5G IoT networks (overlay traffic with several levels of nested encapsulation)	
Measurable Indicator	
Support encapsulation and tunnelling protocols widely used in overlay networks inherent to 4G/5G IoT mobile infrastructures such as VXLAN, GRE, GENEVE or GTP, for instance.	
Baseline value	
In the current state of the art, there is no traffic classifier supporting nested network traffic inherent to 4G/5G and able to perform a Deep Packet Inspection (DPI) of encapsulated traffic where protocols needed for user mobility (GTP) or tenant isolation (VXLAN or GRE). So, the baseline value for this KPI is 0.	
Target value	***Achieved value***
>= 4 (Edge, Core, RAN and Transport)	= 4 (Edge, Core, RAN and Transport)

KPI scope	
OpenFlow protocol extension to provide a flexible and extended programmability of the data plane.	
Measurable Indicator	
% Recovery of the flow services prior to anomalous behaviour. Reduce human intervention to the strictly required, in healing and recovery procedures.	
Baseline value	
N.A. Latest version of Open Flow cannot provide a flexible, extended, and fine-grained data plane programmability in 4G/5G with overlay networks. So, current OF version cannot be used to enforce security policies in the data plane to stop a known attack and restore the network services to the previous status.	
Target value	***Achieved value***
> 95% service recovery Towards 0 % human intervention	100% services successfully recovered. About 0 % human intervention

KPI scope	
OVS Netlink API extension for inter-process communication kernel-user space	
Measurable Indicator	
Number of different flows mitigated Bandwidth performance	
Baseline value	
N.A. In the current SotA, there is no solution provided to stop known attacks when dealing with 4G/5G encapsulated traffic with nested headers for user mobility and tenant isolation.	
Target value	***Achieved value***
Up to 10^6 malicious flows	1048576 (2^{20}) malicious flows when NSP acting like a 4G/5G firewall

Support to work up to 10 Gbps	13.82 Gbps when dealing with 72000 users (IoT devices) connected to the network.

All these KPIs are related to the main project objective "Self and coordinated healing with reduced human intervention". Thus, this component will further contribute to mitigate and reduce the number of cyber incidents involving IoT devices.

3.4.2 Technology research

3.4.2.1 Background

Virtual switches in 4G/5G IoT multi-tenant virtualized infrastructures play a crucial role since this is the place in the network where the physical network managed by Infrastructures Service Providers (ISPs) and the overlay virtual networks built on top of those physical networks intersect. This concrete segment in the Data Plane (DP) pipeline where both types of infrastructures, physical and virtual, converge is named Software Data Path (SDP). The Network Self-Protection entails a novel SDN-based Software Data Plane Agent providing improved programmability beyond the current state of the art to allow the enforcement of security policies aimed at mitigating known DDoS attacks.

As a first step to the design and prototyping of the component, the following features have been identified as needed for the Network Self-protection component to meet the functionality and use cases requirements expected in the project:

- A programmable Software Data Plane agent able to enforce dynamically security policies to mitigate known DDoS attacks in real time.
- Support for protection rules into IoT networks, overlay networks with several levels of nested encapsulations, and any other ARCADIAN-IoT related networks.
- Support for protocols widely used in IoT mobile networks, data centres and cloud deployments like for example:
    - GTP to ensure user and device IoT devices mobility across antennas.
    - VXLAN, GENEVE or GRE to guarantee isolation between tenants in multi-tenancy infrastructures.
- Large scalability to deal with a vast number of IoT devices expected in 5G networks and the ARCADIAN-IoT use cases.

After a comparative analysis of the existing state-of-the-art software data paths, such as DPDK[20], XDP[21] (eXpress Data Path) or OpenVSwitch[22] (OVS) just to mention a few, OVS has been chosen as a baseline for the implementation of the Network Self-protection component. The main reasons motivating this decision are the following:

- OVS is an open-source virtual switch platform supported by a community workgroup keeping it in continuous development.
- It is a well-known software switch widely used in Software Defined Networks (SDNs) and

[20] https://www.dpdk.org/

[21] https://prototype-kernel.readthedocs.io/en/latest/networking/XDP/

[22] https://www.openvswitch.org/

virtualized networks where it is considered the de-facto standard.

- OVS provides a programmable OpenFlow NBI.
- It is widely used for cloud computing frameworks such as OpenStack[23].
- Robustness and satisfactory performance.

The Network Self-protection component is being implemented on OVS version 2.16.2 which can run on Linux kernels from 3.16 to 5.8. We use the latest OpenFlow specification (version 1.5.1 released in 2015).[24]

3.4.2.2 Research findings and achievements

The development of the Network Self-protection component involves a significant extension of the OVS base architecture and functionality aimed at fulfilling the requirements of the project. The following items have been identified during the design stage as extensions to be implemented in the OVS architecture:

- Novel traffic 5G IoT classifier extended from traditional traffic classification for IP networks to a more complex classifier able to deal with all data paths expected in 5G IoT infrastructures such as overlay traffic with several levels of nested encapsulation.
- Increase the expressiveness of the OpenFlow tables managed by the PD subcomponent with new fields providing a flexible, fine-grained flow definition fitting with the new fields extracted by the novel 5G IoT traffic classifier.
- OpenFlow protocol extension aligned the two previous points to provide a flexible fine-grained flow definition, enhanced control, and extended programmability to the Network Self-healing component.
- Extension of the OVS OpenFlow NBI (*ofproto* library) functionality with the OpenFlow protocol extensions described in the previous point allowing the Protection Decider to send and receive extended Open Flow messages from the PCA.
- Extension of the Netlink API for the inter-process communication between the Datapath Security Controller in kernel space and the PD in user space.
- Upgrading of command line application suite included in the OVS distribution with the new capabilities to allow programmability via command line as an alternative method to Open Flow sockets.

It is worthy to further discuss two of these extensions:

- The 5G IoT traffic classifier
- The extensions carried out on the Open Flow protocol.

Regarding the 5G IoT traffic classifier, this module is the entry point of the network traffic into the Software Data Path. This module is part of the Datapath Security Controller sub-component, and it is responsible of performing a deep inspection of the protocol headers of every packet passing through the Software Data Plane. As a result, the classifier extracts information from every packet. This extracted data is kept in a key flow associated to every packet and it is sent to the next module in the Software Data Plane pipeline, the Flow Match Action module. The flow key contains the needed information to allow the Flow Match Action module to take a decision for the packet by matching its associated flow key against the slices enforced in the Data Plane. The final

23 https://www.openstack.org/

24 https://opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf

architectural design of the 5G IoT classifier is provided in Figure 33, where it can be observed the re-entrance between modules responsible for each supported protocol and all the different tagging, tunnelling, overlay, and L2/L3/L4 supported protocols.

When compared to traditional 5-tuple traffic classifiers based on the TCP/IP network stack, the main novelty is that the proposed novel 5G IoT classifier is able to extract information from the inner headers in overlay networks with several levels of nested encapsulations as expected in virtualised multi-tenant 5G infrastructures. This new extended key flow allows a more flexible and fine-grained traffic classification that will enable the enforcement of customized security policies in the data plane to meet the requirements agreed with the ARCADIAN-IoT end-users, verticals, and stakeholders in terms of security and privacy levels.

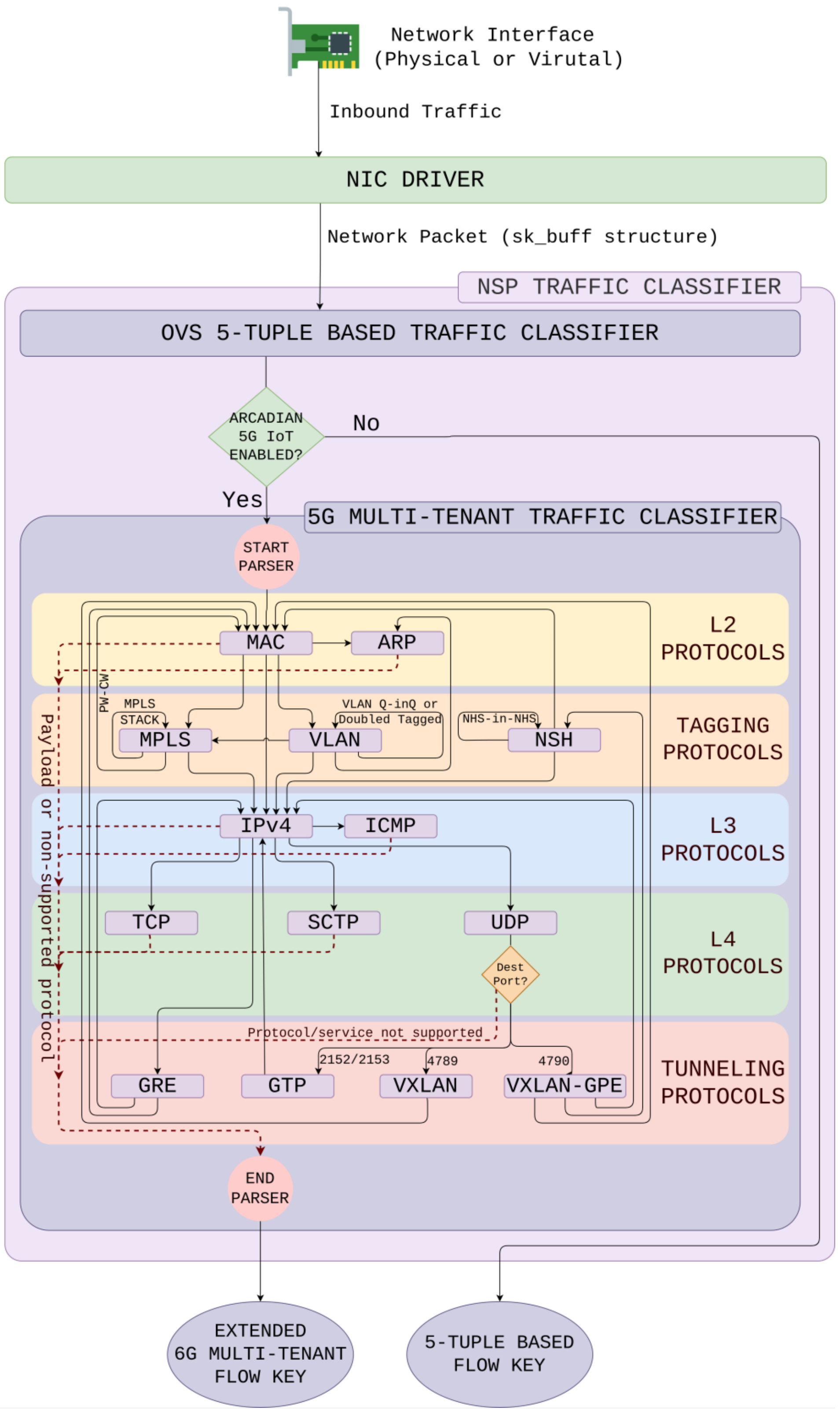


Figure 33 - Overview of the 5G IoT traffic classifier integrated in the Network Self-Protection

Concerning the Open Flow contribution, the protocol has been extended with 17 new fields matching the data in the 5G flow key extracted by the 5G IoT classifier. The new fields contain information from the inner headers allowing to identify IoT devices, services, and users in an unambiguous way. It also provides data about the different encapsulations and tunnels through which IoT traffic traverses the network. Table 12 lists the proposed new fields along with the length in bytes and the field description.

Open Flow Field	Length (Bytes)	Description
inner_ip_src	4	Source IP address in the inner IP4 header
inner_ip_dst	4	Destination IP address in the inner IPv4 header
inner_tos	1	ToS/DSCP value in the inner IPv4 header
inner_port_src	2	Source port in the inner UDP header
inner_port_dst	2	Destination port in the inner UDP header
inner_l3_protocol	2	Ethernet type of the inner layer 3 header
inner_l4_protocol	1	IANA protocol number of the inner layer 4 header
tunnel_type_1	1	Tunnel type of encapsulation level 1
tunnel_type_2	1	Tunnel type of encapsulation level 2
tunnel_type_3	1	Tunnel type of encapsulation level 3
tunnel_type_4	1	Tunnel type of encapsulation level 4
tunnel_type_5	1	Tunnel type of encapsulation level 5
tunnel_key_1	4	Tunnel key of encapsulation level 1
tunnel_key_2	4	Tunnel key of encapsulation level 2
tunnel_key_3	4	Tunnel key of encapsulation level 3
tunnel_key_4	4	Tunnel key of encapsulation level 4
tunnel_key_5	4	Tunnel key of encapsulation level 5

Table 12 - Proposed new Open Flow fields

It is worth highlighting that the protocol header field matching the tunnel ID depends on each encapsulation protocol context as indicated in Table 13. For example, in the case of GTP, used to deliver IoT device and user mobility across the antennas and the network, the tunnel ID refers to the Tunnel Endpoint Identifier (TEID) field in the GTP header. In VXLAN, used to guarantee isolation between different tenants sharing the same physical infrastructure, the tunnel ID corresponds to the VXLAN Network Identifier (VNI) field of the VXLAN header. Notice that the pair (VNI, TEID) identifies unequivocally a IoT device.

Encapsulation Protocol	Tunnel type value	Tunnel key description
VXLAN	1	VXLAN Network ID (VNI)
GPT	2	Tunnel Endpoint Identifier (TEID)
GRE	3	GRE Key
MPLS	4	MPLS Label
VLAN	5	VLAN Identifier (VNI)
NSH	6	Service Path Identifier (SPI)
VXLAN-GPE	7	VXLAN Network ID (VNI)
GENEVE	8	Virtual Network Identifier (VNI)

Table 13 - Parsing of encapsulation, tunnelling & tagging protocols: tunnel type values & key descriptions

3.4.2.3 Produced resources

Due to the application of IPR safeguarding measures, UWS has not the intention to make any release of the source code for this component and thus it will not be made available in any public or private repository or server outside of our premises. For integration purposes, a functional prototype of the Network Self-Protection component will be released for the ARCADIAN-IoT consortium.

3.4.3 Design specification

3.4.3.1 Logical architecture view

The overall view of the Network Self-protection architecture designed by UWS is provided in Figure 34, which highlights the three main subcomponents integrating the functionalities previously described: Protection Control Agent (PCA), Protection Decider (PD) and Datapath Security Controller (DPSC). The internal and external interfaces are depicted in the figure as well (a functional description of the interfaces is available in deliverable D2.5.

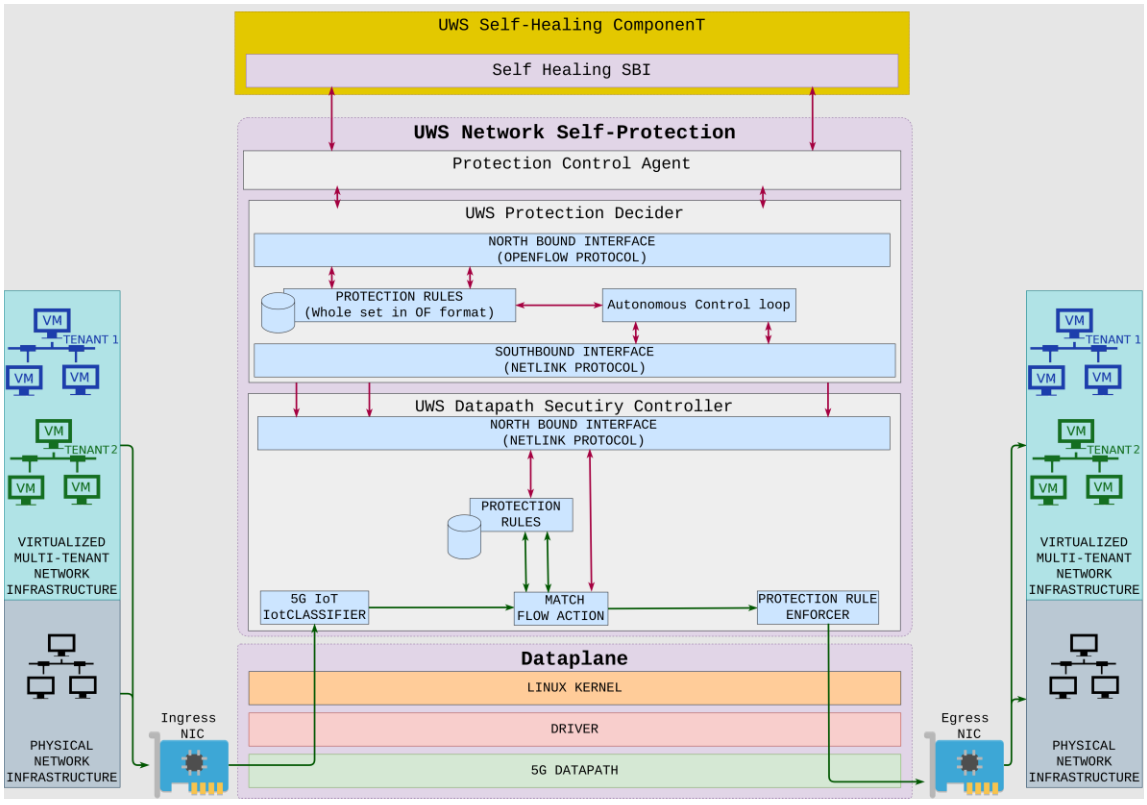

UWS Self-Healing ComponenT
Self Healing SBI
UWS Network Self-Protection
Protection Control Agent
UWS Protection Decider
NORTH BOUND INTERFACE (OPENFLOW PROTOCOL)
PROTECTION RULES (Whole set in OF format)
Autonomous Control loop
SOUTHBOUND INTERFACE (NETLINK PROTOCOL)
UWS Datapath Secutiry Controller
NORTH BOUND INTERFACE (NETLINK PROTOCOL)
PROTECTION RULES
5G IoT IotCLASSIFIER
MATCH FLOW ACTION
PROTECTION RULE ENFORCER
Dataplane
LINUX KERNEL
DRIVER
5G DATAPATH
Ingress NIC
Egress NIC
VM
TENANT 1
TENANT 2
VIRTUALIZED MULTI-TENANT NETWORK INFRASTRUCTURE
PHYSICAL NETWORK INFRASTRUCTURE

Figure 34 - Network Self-Protection architectural overview

The functionality of the subcomponents is described as follows:

- The self-management capabilities are achieved by the **Protection Decider (PD)** which is responsible for deciding on every instant what subset of rules from the complete set of protection rules located in the PD will be enforced in the DSC by creating an autonomous control loop to perform self-optimization of the protection capabilities and thus achieving large scalability, really needed to deal with security on IoT networks. The PD is continuously monitoring the behaviour of the DPSC to optimise its performance.
- The **Datapath Security Controller (DPSC)** is the subcomponent that processes the

network traffic and thus eventually enforces the protection rules into the data plane. Every packet through the data plane is deep inspected and classified, and an action is taken based on the subset of protection rules active at the moment. If a packet does not match any rule, it is sent to the PD subcomponent so the subset of protection rules kept in the DPSC tables can be updated.

- Finally, the third subcomponent is the **Protection Control Agent (PCA).** It is responsible of providing a North Bound Interface (NBI) for the communication with the Network Self-healing component. The exposed NBI is an intend-based interface receiving technology-independent instructions that are translated into technology dependant commands to be enforced in the data plane. The PCA subcomponent allows dynamic management of the life cycle of the set of protection rules in the self-management system: installation, modification, and deletion.

3.4.3.2 Sequence diagram

The sequence diagram showed in Figure 35 illustrates the interaction between the 3 ARCADIAN-IoT components composing the cognitive loop deployed in the Edge and Core segments of the networks. These 3 components collaborate together to protect the network and therefore the ARCADIAN-IoT infrastructure. The Network Flow monitoring detects illegitimate traffic coming from a cyber-attack and raises an alert as a Network IDS Event (1 in Figure 35). This alert is consumed and processed by the Network Self-Healing component (2 in Figure 35). The NSH, based on the network topological and resources information gathered, takes a decision on how to stop the attack: At which point of the data plane pipeline, when to stop the attack and how to stop the attack. The healing decision is sent in form of Self-Healing instruction to the RabbitMQ exchange where is consumed by the NSP agent (3 and 4 in Figure 35). The NSP is the ARCADIAN-IoT component which is eventually enforcing the security policies in form of self-protection rules in the data plane to mitigate the attack.

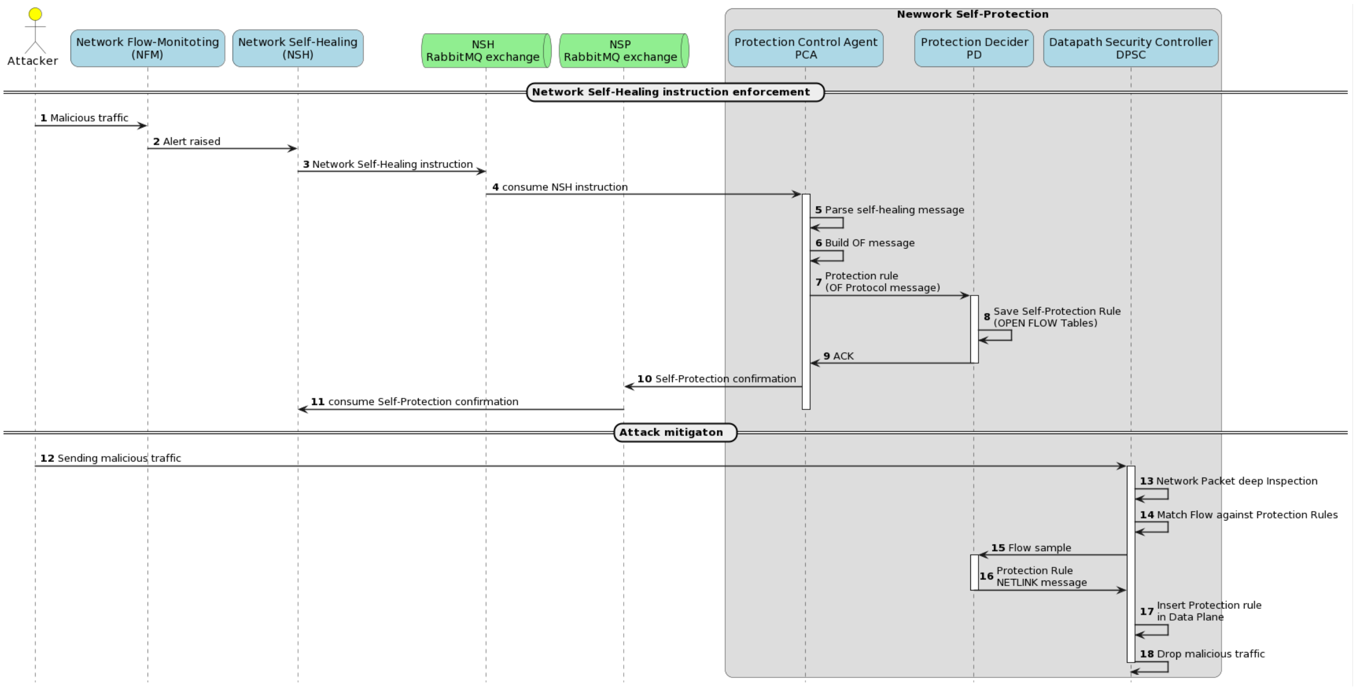

Attacker
Network Flow-Monitoring (NFM)
Network Self-Healing (NSH)
NSH RabbitMQ exchange
NSP RabbitMQ exchange
Newwork Self-Protection
Protection Control Agent PCA
Protection Decider PD
Datapath Security Controller DPSC
Network Self-Healing instruction enforcement
1 Malicious traffic
2 Alert raised
3 Network Self-Healing instruction
4 consume NSH instruction
5 Parse self-healing message
6 Build OF message
7 Protection rule (OF Protocol message)
8 Save Self-Protection Rule (OPEN FLOW Tables)
9 ACK
10 Self-Protection confirmation
11 consume Self-Protection confirmation
Attack mitigaton
12 Sending malicious traffic
13 Network Packet deep Inspection
14 Match Flow against Protection Rules
15 Flow sample
16 Protection Rule NETLINK message
17 Insert Protection rule in Data Plane
18 Drop malicious traffic

Figure 35 – Sequence Diagram describing interaction between Network Self-Protection and other ARCADIAN components involved in the Self-Healing loop

The PCA, that is the Northbound API sub-component of the NSP, parses the message sent by the NSH and build an Open Flow message containing a self-protection rule that is sent to the PD (5, 6 and 7 in Figure 35). The PD save the rule in user space in Open Flow format, but the rule is not yet applied in the network traffic pipeline. This is mechanism to optimise the performance of the DPSC when managing traffic. Once the self-protection rule has been inserted in the Open Flow tables (8 in Figure 35), the PD sends an ACK message to the PCA which in turn confirms to the NSH that the self-healing instruction has been successfully applied (9, 10 and 11 in Figure 35). When the first packet of malicious traffic arrives at the DPSC instance, it is parsed, and the flow key extracted is matched against its protection rules. Unlike the PD which is a user-space daemon, the DPSC is a kernel module, and the self-protection rules are not stored in Open Flow format but in binary format similar to the flow key extracted by the 5G IoT classifier. This allows for higher performance in handling network traffic, thus providing improved bandwidth. The PD is continuously monitoring the subset of Open Flow rules inserted into the DPSC tables, optimising it by removing the unnecessary rules (e.g., rules not matching traffic since n seconds). Since there is no rule matching the packet in the DPSC, the packet is sent to the PD (15 in Figure 35). The PD sends to the DPSC the self-protection rule to be inserted in the data plane (16 in Figure 35). The communication between the PD and the DPSC subcomponents is accomplished by using the NETLINK inter-process communication. Once the self-protection rule inserted into the data plane (DPSC rule table) (17 in Figure 35), all harmful traffic matching the rule will be dropped, resulting in stopping the attack at the desired point in the network (18 in Figure 35).

3.4.3.3 Interface description

- Network Healing Instructions is the interface that the NSH component will use to provide the Network Self Protection component with the necessary information about what, where and how to mitigate the threat happening in the detected network segment.
- Network Self-Protection confirmations is the interface that the Network Self-Protection is using to provide information about the successful enforcement of the security rule provided by the Network Self-Healing component.

Partner	Definition	Channel	Exchange
UWS	Network Self-Healing	RabbitMQ AMQP 0.9.1	Network Self-Healing Instructions
UWS	Network Flow Monitoring	RabbitMQ AMQP 0.9.1	Network Self-Protection Confirmations
UC	Reputation System	RabbitMQ AMQP 0.9.1	Network Self-Protection Confirmations

Table 14 - The external interfaces of the Network Self-protection component

3.4.3.4 Technical solution

3.4.3.4.1 Deployment architecture view

Figure 36 denotes the global deployment architecture view followed by the UWS cognitive loop for network security components alongside an entire 5G network infrastructure. The 5G architecture is composed by different network segments and layers. It can be differentiated the Radio Access Network (RAN) segment, where IoT devices are connecting to the different 5G gNB with its respective Distributed Unit (DU) that connects to the consequent network segment, the Edge. This network segment oversees the responsibility of deploying and managing the digital services from the rest of the stakeholders involved in the system, as Digital Communication Services Providers. They use their virtual Central Units to manage communications across the rest of the infrastructure. In addition, in the Edge segment can be found Multi-access Edge Computing applications in order to approach computing services and capabilities to the end user. The transport network segment is the one between the Edge and Core premises, where the traffic is routed from the specific geographically distributed Edge to the centralized Core. The last network segment in the infrastructure is the Core, where first user authentication takes place. Once the communications leave the Core segment, they are forwarded to an inter-domain network segment outside the infrastructure provider's domain.

The Network Self-protection component is deployed in the software data path as an SDN-based virtual software switch. It is deployed as a distributed component in each one of the machines within the cloud infrastructure where the cyber-attack could be mitigated.

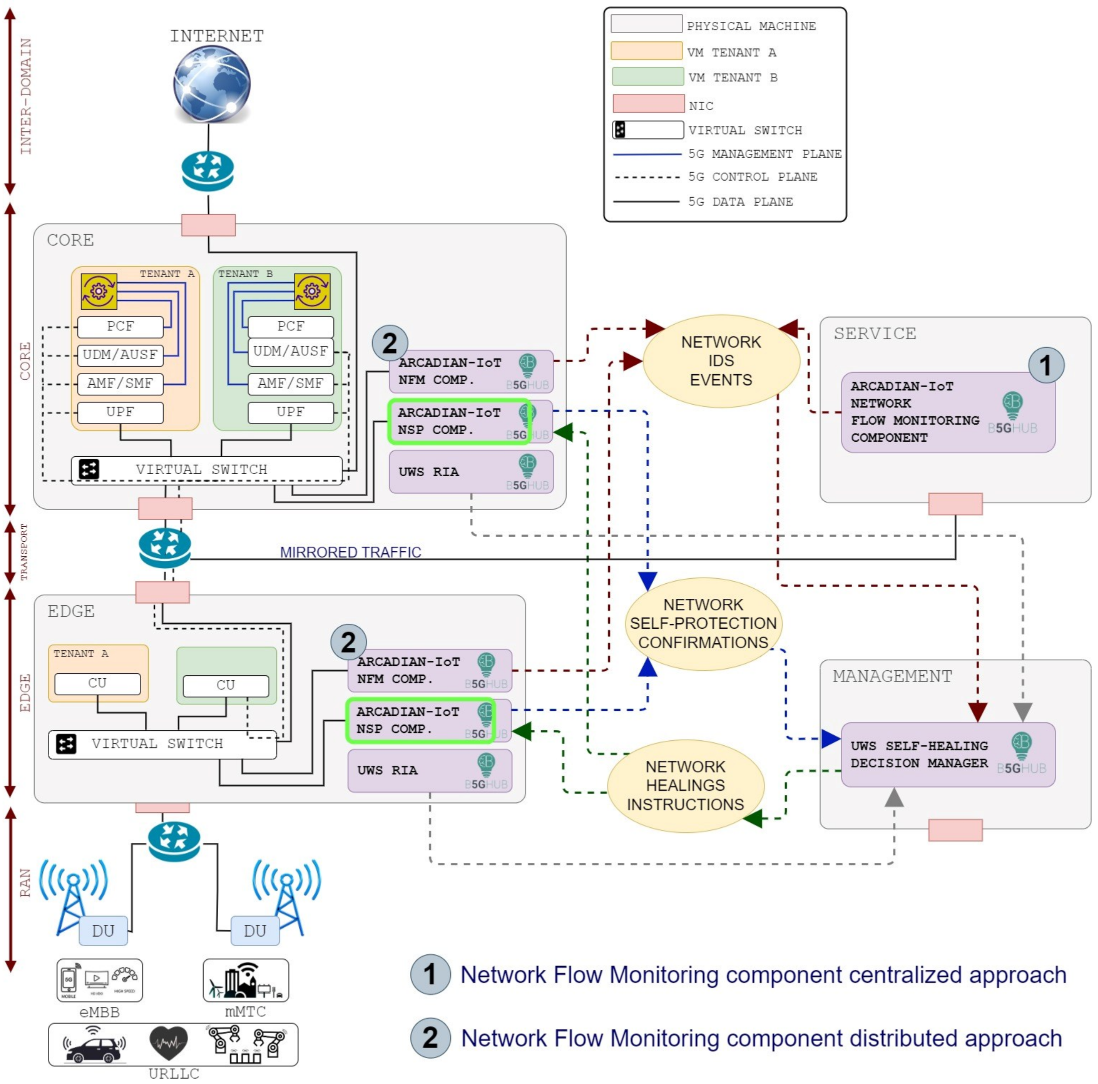


Figure 36 - Network Self-Protection Loop architectural overview with NFM, NSH and NSP integrated.

3.4.3.4.2 *API specification*

The Network Self-Protection component is using a message bus publication/subscription channel to interact with the rest of the components in the ARCADIAN-IoT framework. This component uses two different exchanges: the first is the Network Self-Healing Instructions, where the Network Self-Healing will be publishing the specific healing instructions with information about the enforcement action that might be taken by the NSP component. As a second interface, the NSP component will be publishing to the Network Self-Protection confirmations. This information will be used to confirm the security rule has been already enforced by the NSP component and thus the effectiveness of the cognitive self-protection security network loop will be also confirmed.

3.4.4 Evaluation and results

The Network Self-Protection ARCADIAN–IoT component was validated in the same testbed described in section 3.1.4 for the Network Flow Monitoring (NFM). This testbed integrates the 3 architectural components conforming the Network Self-Healing Loop, that is, the NFM, the Network Self-Healing and the Network Self-Protection. Thus, the testbed configuration and different scenarios shown in Table 4 are the same for the 3 components. Section 3.1.4 provides a complete description and discussion of the testbed.
In examining the NSP component behaviour in Figure 20 in section 3.1.4, it becomes apparent that it exhibits a longer time consumption in scenario A compared to scenario B, although it maintains consistency across all settings within each scenario. Specifically, NSP required approximately 5 seconds in scenario A and roughly 1.5 seconds in scenario B to execute the healing directives within the data plane and effectively terminate the attack. The variance in response times for NSP arises from its distributed deployment. This component is strategically distributed across the various compute nodes within the network topology, resulting in a workload distribution that aligns with the number of NSP instances deployed throughout the infrastructure. Capitalizing on this distributed nature, Figure 20 illustrates improved performance in NSP as the network topology expands, even as the number of IoT devices remain constant. This underscores the benefits of leveraging the distributed architecture, particularly when network complexity increases.

Furthermore, it is essential to emphasize that all the configurations outlined in 4 encompass scenarios and topologies characterized by the involvement of multiple stakeholders, the incorporation of diverse network segments, and the utilization of various overlay network layers within the network architecture. Consequently, the KPIs for the NSP component have been comprehensively addressed and successfully attained across these experimental scenarios.

3.4.5 Future work

A prospective direction for future research, beyond the ARCADIAN-IoT, involves delving into the integration of the proposed framework with multiple kernel bypass technologies. This integration would extend to include technologies such as the extended-OVS (Open vSwitch) kernel module, Data Plane Development Kit (DPK), and eXpress. Such an exploration would enhance the framework's capacity to leverage cutting-edge technologies for optimizing network performance and security.

3.5 IoT Device Self-protection

3.5.1 Overview

3.5.1.1 Description

The Device Self Protection is an ARCADIAN-IoT component designed to protect IoT devices from threats or anomalies detected by other components of the ARCADIAN-IoT framework. The IoT Device Self-Protection (DSP) component enforces protection policies at the device level whenever possible and suggests policies to apply to when direct application is not possible (e.g., due to business logic rules or device restrictions). These policies include disabling access to specific network protocols or preventing access sensitive data or directories in the device. Another objective of this component is to reduce human intervention, with the indication of appropriate protective measures and for the implementation of automated rule enforcement.

To perform these actions, the IoT Device Self-Protection relies on information derived from other ARCADIAN-IoT components such as the Device Behaviour Monitoring (DBM), Cyber Threat Intelligence (CTI) or Reputation System (RS). These provide information on detected anomalies, indicators of compromise and reputation levels, respectively. These inputs, when combined with tailored risk levels, allow the creation of a greater variety of policies and thus better protection of IoT devices.

After analysing the inputs that are provided to DSP component, the application of policies to vulnerable devices is either directly applied or suggested (via RabbitMQ queues) to domain administrators. Information about policy enforcement is announced to the Reputation System, and when applicable, to the Self-Recovery component – which may trigger recovery actions or preventive measures with respect to data backup.

3.5.1.2 Requirements

A recall of the high-level requirements that have been previously defined and provided in deliverable D2.4 is given below.

Requirement 3.5.1 - Heartbeat monitoring mechanisms towards ARCADIAN-IoT framework should be implemented.

Requirement 3.5.2 - Device should allow at least one type of adaptive settings (e.g., dynamic configuration, rule enforcement, permission granting/revoking).

Requirement 3.5.3 - Device should provide administrative privileges to self-protection component.

Requirement 3.5.4 - Device should be able to periodically obtain up-to-date classification of applications and services (e.g., from reputation system or CTI).

3.5.1.3 Objectives and KPIs

KPI scope	
Policy enforcement based on ensemble of risk levels, indicators of compromise, reputation and threat information	
Measurable Indicator	
Number of policy enforcement methods.	
Benchmarking	
Support at least a risk level protection approach	
Target value	***Achieved value***
Support at least 3 policy enforcement methods (e.g., risk levels, reputation information, threat or indicators of compromise).	Supporting 3 policy enforcement method (i.e., threat information for intrusion detection system, reputation system and indicators of compromise).

KPI scope	
Autonomous self-protection mechanism for heterogeneous operating systems and devices.	
Measurable Indicator	
Type of supported operating systems (e.g., Linux and android) and type of device (e.g., drones and smartphones).	
Target value	***Achieved value***
Support at least 2 Operating Systems and 2 device types	Supporting 2 Operating System (Linux and Android) and 2 device types (drone and

	smartphone).

3.5.2 Technology research

This section gives background information on the problem, highlighting research that has been conducted with respect to state-of-the-art in Self-Protection solutions, as well as research findings and resources that have been created.

3.5.2.1 Background

The growth in the number of IoT devices in recent years has forced developers to pay more attention to security issues. The greater the variability of IoT devices, the greater the diversity of threats. According to OWASP (The Open Web Application Security Project), which periodically publishes the top 10 causes of device security failures, the main problems are weak passwords, insufficient privacy protection or lack of proper device management [52].

It is essential to enforce a security layer capable of reducing the effectiveness of attacks and intrusions attempts on IoT devices. Self-protection mechanisms are designed to increase runtime software system security. By having inputs on the status of a system and its execution environment, self-protection mechanisms can automatically prevent or minimize security risks by making decisions based on the observed state and implementing enhanced security measures that are appropriate for the present threat. The advantage of these mechanisms is that they enable the protection of IoT devices with reduced or no human intervention.

This work also required the identification of self-preservation methodologies related to those slated for incorporation within the ARCADIAN-IoT framework, while concurrently stimulating enhancements for the targeted component. The initial step considered the pursuit of an architectural framework aligned with the specific objectives of device self-protection component. In this context, the MAPE-K architecture emerged as particularly noteworthy, given its segmentation of the system's operation into the sequential stages of:

- Monitor: collects data from managed element.
- Analyzer: examines the data collected by the Monitor and determines whether any variations relevant happened.
- Planner: determines which system modifications should be made to meet the variances found by the Analyzer and specifies the set of operations to be used in the system.
- Executer: applies the system changes recommended by the Planner.
- Knowledge Base: includes data from the Monitors that have been filtered, statistical data, and self-protection strategies that have been used in past instances.

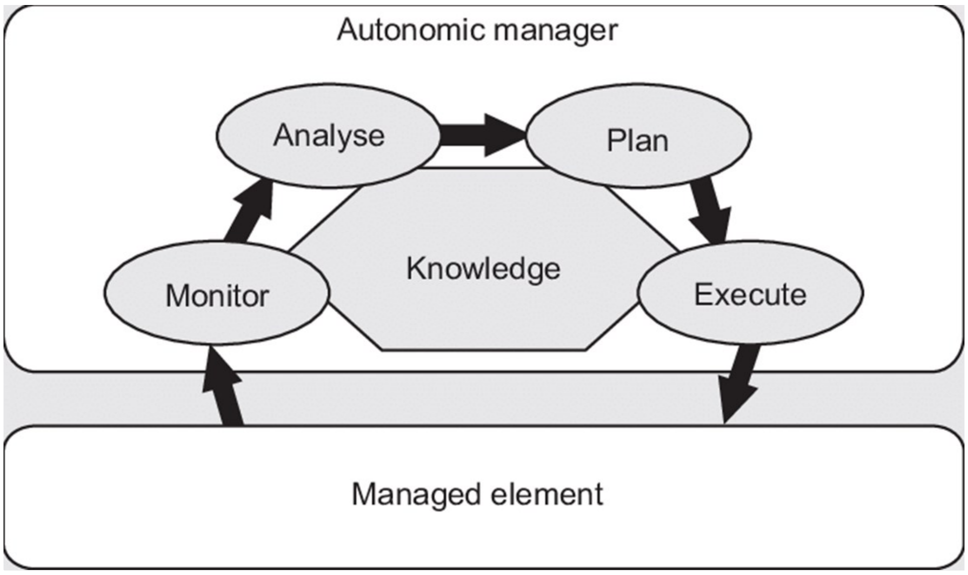


Figure 37 - MAPE-K Architecture Representation [53]

The architecture represented in Figure 37 depicts the logical architecture of protection loop, in which the IoT Device Self Protection component is based. The Monitor corresponds to the module responsible for receiving new alerts from other ARCADIAN-IoT components (e.g., Device Behaviour Monitor or Device Self Protection. The Analyse and Planner blocks correspond to modules that deal with data processing, finding the best response (i.e., policy) to a threat. The Executer block corresponds component module which will enforce or suggest the recommend policies. Finally, the Knowledge Base holds the policies that may be applied to each of the detected threats.

3.5.2.2 Research findings and achievements

During the research process, we sought alternatives for data analysis beyond the examination of data provided by the components feeding into IoT Device Self-Protection. One identified approach relates to enable protective responses based on risk levels. For instance, consider a straightforward scenario: a failed attempt to log in to an application may be deemed a low-risk event. However, if the frequency of such attempts is notably high, the potential for suspicious activity proportionally increases, prompting consideration of additional protective measures. In the first scenario, the failed attempt may be considered inconsequential; however, in the latter case, it could imply a brute force attack, allowing the implementation of measures such as a request for secondary authentication. This approach enables a broader coverage of actions, thus augmenting the layers of security for the devices involved.

Ultimately, an additional capability concerning IoT Device Self-Protection pertained to the methodology for preserving device protection policies. The consideration of employing a database was dismissed, given its inherent complexity, which would prove heavy should this component be installed on devices, as it is the case within ARCADIAN-IoT. Moreover, database access typically incurs latency, potentially resulting in performance degradation. The identified resolution revolves around the utilization of the OPA policy engine. This tool facilitates the creation and archival of new policies in a *.rego* file format. Consequently, not only does this approach ensure the efficiency of the component, but it also streamlines the processes of policy composition and retrieval.

Overall, the development of the IoT Device Self-Protection component has considered the following aspects:

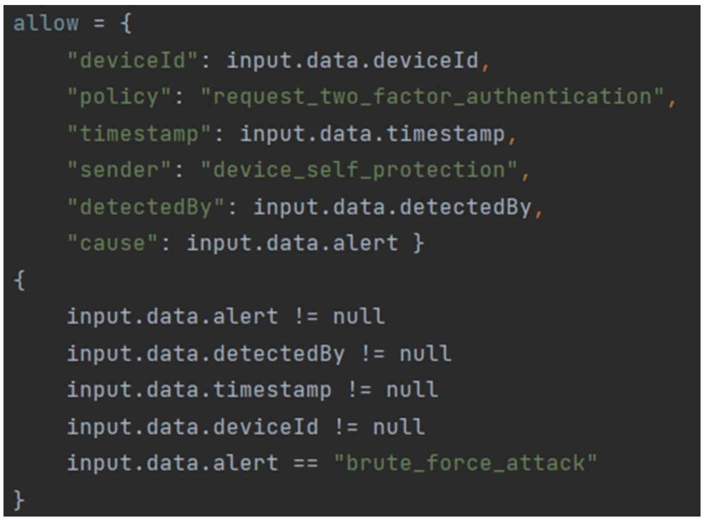
- **Policy Storage:**
  - Another functionality is the ability to store self-protection policies (i.e., actions and rules) via a third-party policy manager - Open Policy Agent (OPA). This allows the component to store several self-protection policies in an efficient and structured manner.
  - The policies may include an enumeration of actions to perform to apply certain policies. For instance, the policy "Block incoming SSH communications" includes actionable information that the device must DROP incoming traffic on port 22 or DROP all incoming traffic identified with the SSH protocol.
  - The device self-protection component supports (in the use cases where it is applicable) the automatic enforcement of protection policies directly on the device. This enables the component to make changes on the device by translating the protection policy in actionable commands (e.g., dropping incoming traffic on port 22 requires making changes on the device's firewall to apply such policy).
  - In the cases where automatic rule enforcement is not available, this component makes the recommended actions available to the system managers or administrators via the RabbitMQ exchange.

- **Hybrid Deployment Method:**
  - As previously mentioned, the Device Self Protection component works with a local OPA version (via binary/executable) [55]. This version deploys on Linux based devices, which means that the policies can be obtained locally via a shell command.
  - Alternatively, it also deploys in server mode, where an OPA container is deployed remotely with the desired policies, and thus, the client can make a request with a specific input to the server and receive a response with the respective policy to apply.
- **Policy enforcement methods:**
  - The current policies were defined according to the specificities of the use cases and can also be adapted to encompass other inputs from other components.
  - The current implemented policies are listed in the table as follows:

Policy Name	Sender Component	Description/Action
close_ssh_port	Device Behaviour Monitoring	Drops traffic on port 22 via the iptables rules. Applied when an intrusion is detected on the sshd service.
close_ftp_port	Device Behaviour Monitoring	Drops traffic on port 21 via the iptables rules. Applied when an intrusion is detected on the ftp service.
request_two_factor _authentication	Device Behaviour Monitoring	Applied when an anomaly is detected on authentication events.
sudden_reputation_score_ change_warning	Device Behaviour Monitoring	Restrict device user access of specific directories (e.g., user folder/system folder).

		Applied when a reputation score of reputation is low enough.
lower_reputation _score_warning	Reputation System	Restrict device user access of specific directories (e.g., user folder/system folder). Applied when a reputation score of reputation is low enough.
low_reputation_alert	Reputation System	Delete compromised user. Applied when a extreme low score of reputation is detected.
recover_high_reputation	Reputation System	Reverse device user access of specific directories (e.g., user folder/system folder). Applied when a reputation score of reputation is high enough.
sudden_reputation _score_change_warning	Device Behaviour Monitoring	Restrict device user access of specific directories (e.g., user folder/system folder). Applied when a reputation score of reputation is low enough.
lower_reputation_ score_warning	Reputation System	Restrict device user access of specific directories (e.g., user folder/system folder). Applied when a reputation score of reputation is low enough.
user_threat	CTI	Delete user/ Restrict device user access of specific directories (e.g., user folder/system folder). Applied when a IoC is shared by CTI with an user entity that presents a threat.
malware_threat	CTI	Restrict device user access of specific directories (e.g., user folder/system folder). Applied when a IoC is shared by CTI with a malicious file/folder that presents a threat.

Table 15 - Applicable policies and descriptions



```
allow = {
    "deviceId": input.data.deviceId,
    "policy": "request_two_factor_authentication",
    "timestamp": input.data.timestamp,
    "sender": "device_self_protection",
    "detectedBy": input.data.detectedBy,
    "cause": input.data.alert }
{
    input.data.alert != null
    input.data.detectedBy != null
    input.data.timestamp != null
    input.data.deviceId != null
    input.data.alert == "brute_force_attack"
}
```

Figure 38 - Example of the policy file structure of Device Self Protection using Open Policy Agent

3.5.2.3 Produced resources

The developed code is written in Python 3 and is available on the project's GitLab repository [54].

3.5.3 Design specification

The logical architecture view is considered in this section, where we explore the internal and external interfaces of the component, sequence diagrams, the deployment architecture model, interface definition, and the resulting technical solution.

3.5.3.1 Logical architecture view

Figure 39 illustrates the logical architecture that represents the IoT Device Self-Protection component, as well as its internal and external interfaces.

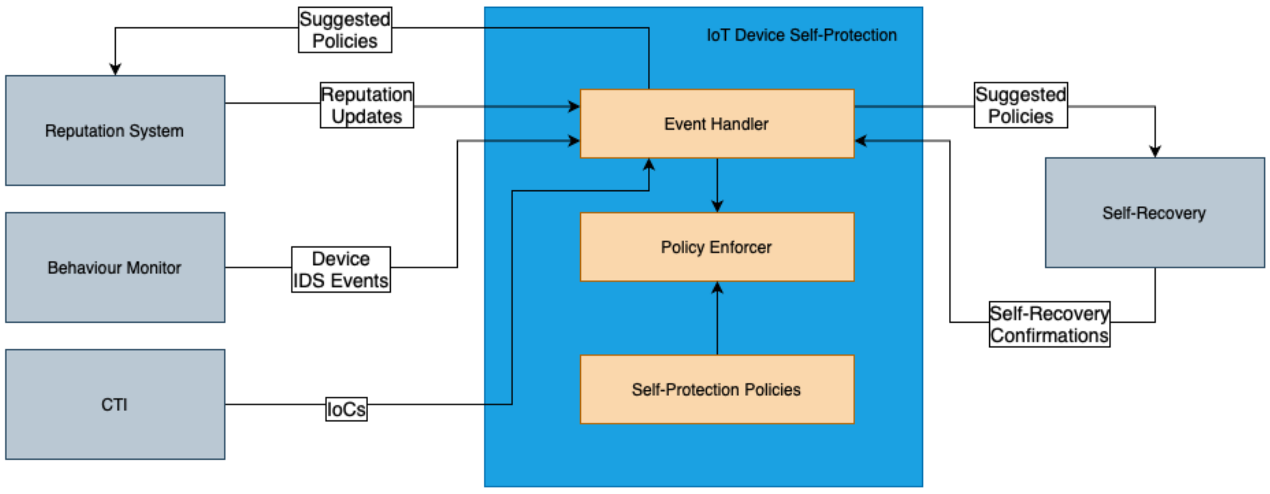


Figure 39 - Device Self-Protection component architecture

The internal workflow is composed of the following subcomponents (internal interfaces):

- Event handler: This sub-component handles incoming events from other ARCADIAN-IoT components like the Behaviour monitor component (e.g., threat/intrusion detection warnings), CTI (e.g., Indicators of Compromise or external threat/intrusion-related information) and Reputation System (e.g., device reputation information). It also translates and forwards the incoming information towards the internal policy enforcer sub-component.

- Policy enforcer: It oversees the application/suggestion of specific protective actions on the device (e.g., revoking permissions and drop incoming/outgoing traffic connections on specific protocols). Based on the incoming events information, the Policy enforcer probes the self-protection policies and selects the most appropriate actions based on the available information.
- Self-protection policies: It is represented by a *.rego* file with the specification of self-protection policies. This sub-component leverages and interfaces with Open Policy Agent (OPA) – described in the following sections – to have policy a knowledge base.

3.5.3.2 Sequence diagrams

The sequence diagram in Figure 40 depicts the information flow of the IoT Device Self Protection. All the actions are initiated from another ARCADIAN-IoT components. The IoT Device Self-Protection receives intrusion alerts, reputation information or indicators of compromise. The Event Handler is responsible for processing and forwarding the incoming information to the policy enforcer, which queries the policy registry to determine the most suitable policy to consider. The Policy enforcer is responsible for recommending an action to be applied on the devices.

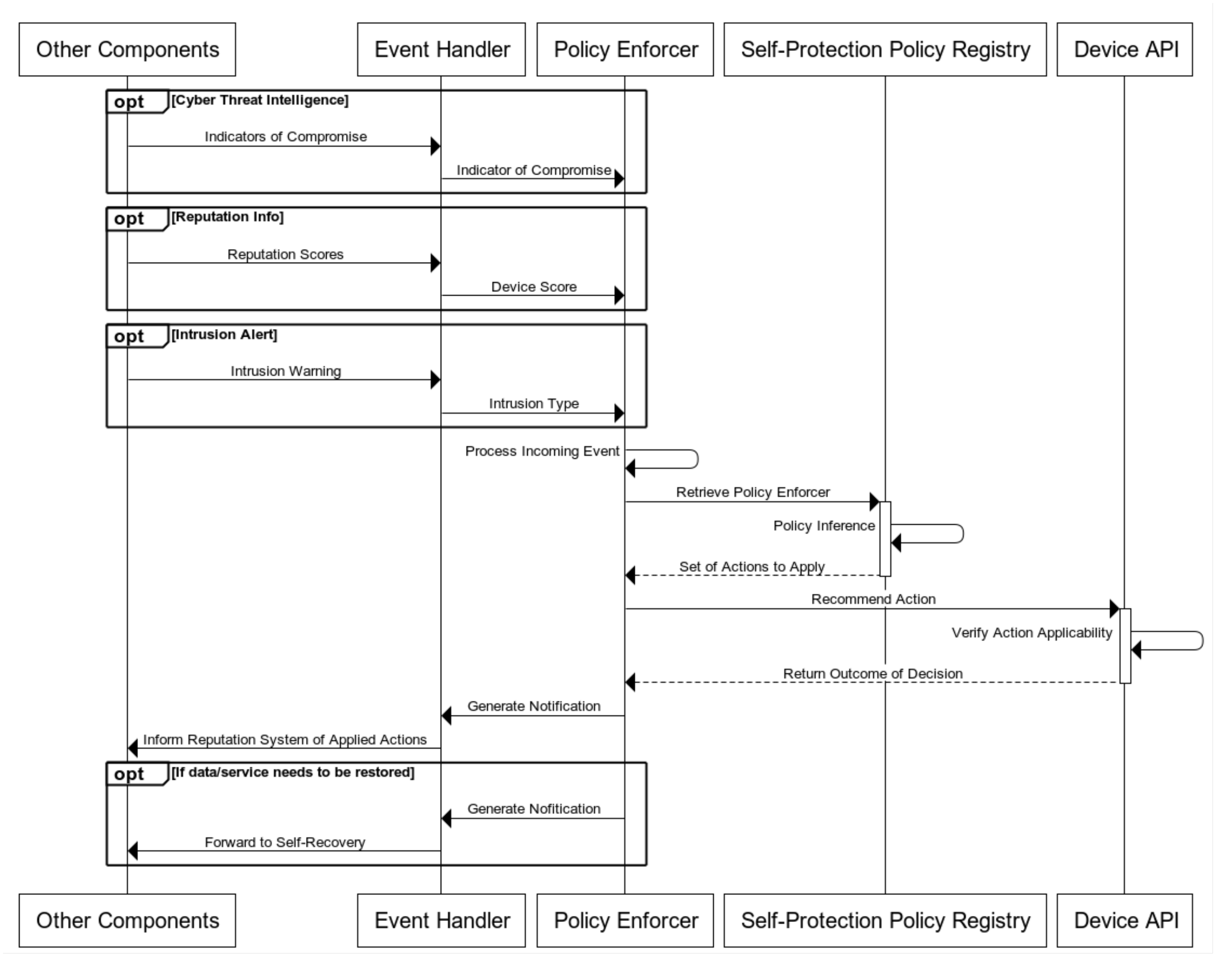


Figure 40 - High level sequence diagram of the IoT Device Self Protection

3.5.3.3 Interface description

The external components that communicate with the device self-protection component are the following:

- Behaviour Monitoring: This interface enables the Device Self-Protection to receive threat/intrusion detection warnings from the Behaviour Monitoring component.
- CTI: This interface allows Device Self-Protection to receive indicators of compromise or external threat/intrusion-related information from the Cyber Threat Intelligence component.
- Reputation System: This interface enables the DSP to receive reputation updates from the Reputation System and allows it to inform the Reputation System about the application of the suggested security policies.
- Self-Recovery: This interface enables device self-protection to send suggested policies to the Self-Recovery component, so that it can initiate data recovery or stop regular data backups. It also allows for receiving confirmations about the application of the policies from Self-Recovery, which it then forwards to the Reputation System.

The component supports communication with ARCADIAN-IoT messages bus (i.e., RabbitMQ). This functionality allows the device self-protection to receive threat alerts from components such as Device Behaviour Monitoring, reputation scores from the Reputation System and Indicators of Compromise from the CTI. Moreover, this functionality allows the component to forward policy enforcement confirmations to other components (e.g., reputation system or self-recovery). This messaging system enables the components to exchange information in standard formats. Additionally, in the use cases where it applies, the device self-protection has the capability to send suggestions via a specialized RabbitMQ queue with the corresponding policy to apply.

The following table summarizes the integration with each of the external components:

Partner	Component	Channel	Exchange
IPN	Behaviour Monitoring	RabbitMQ AMQP 0.9.1	Threat/intrusion detection warnings
UC	Reputation System	RabbitMQ AMQP 0.9.1	Reputation updates
RISE	Cyber Threat Intelligence	RabbitMQ AMQP 0.9.1	Indicators of Compromise
XLAB	Self-Recovery	RabbitMQ AMQP 0.9.1	Service Recovery and Self-Recovery confirmations

Table 16 – IoT Device Self-Protection External Interfaces

3.5.3.4 Technical solution

3.5.3.4.1 API specification

The IoT Device Self-Protection component employs a RabbitMQ publish/subscribe model for its communications with other ARCADIAN-IoT components. It both receives and sends messages. In particular, it sends messages to the Reputation System and Self-Recovery components. The content of these messages varies based on the type of attack, corresponding security policy, and context where the message will be used. The following table presents these parameters in more detail.

Message Format	Parameter	Type	Description	Format
JSON	timestamp	string	Current system time	Unix Timestamp (in seconds) (e.g., 1666620456)
	attack_start_date	string	Time when attack was first detected	Unix Timestamp (in seconds) (e.g., 1666620456)
	attack_end_date	string	Time when attack ended	Unix Timestamp (in seconds) (e.g., 1666620456)
	deviceId	string	arcadian_iot_ID of the device	Source:ID (e.g., ipn:3a45d35a-2494-407b-b8c9-adf39834a38f
	cause	string	Program/Service that was attacked	Service name (e.g., SSH, FTP, etc)
	policy	string	Measure to mitigate the threat	Policy name (e.g., two_factor_authentication)
	processId	string	Process ID associated to the program	ID number of the process (e.g., 9346)
	currentScore	float	Current reputation score of the device	A real number between, and including, 0 and 1
	previousScore	float	Previous reputation score of the device	A real number between, and including, 0 and 1
	detectedBy	string	The component that detected the threat	Name of the component.
	sender	string	Identification of sender component	<domain> <component> <location> <entity_id> <service>
	policy_application	string	Indicates whether the security policy is applied or not.	Boolean value, in string format (true or false)
	policy_enforcer	string	Name of the component that tried to apply the policy.	Either “device_self_protection” or “self_recovery”

Table 17 - Device Self-Protection Output Specification

3.5.3.4.2 *Security aspects*

The execution of the IoT Device Self Protection component will inherently increase the security level of the devices where it is being deployed by actively and passively applying/suggesting protection measures in case of anomalies.

Actions that result from the application of protection policies do not hamper the core functionality of the devices. For instance, the deployment of the component in a drone does not affect its flight control mechanisms – in such scenarios, more restrictive policies are recommended to the business logic.

Additionally, the communication of the IoT Device Self-Protection component with the framework’s RabbitMQ is authenticated. In future releases, additional security layers can be implemented (e.g., TLS certificate usage).

3.5.4 Evaluation and results

Even though this component does not have experimental results like other component (e.g., device behaviour monitoring and its ML models), the implemented functionalities were tests and validated in a standalone mode. The first tests carried out focused on internally testing and validating the IoT Device Self-Protection implementation. Considering the messages generated, the component was always able to return the expected results on each of the queried functions.

Additionally, a complete functional flow was also a target of test and validation. Starting from reading the messages coming from the message bus, performing the necessary parsing and validation of the contents, going through the selection of policies and finalizing with the application/suggestion of the specific policy for the anomalous event detected on the device.

The successful result of these activities provided additional confidence in the robustness and functionality of the component. In addition, these results facilitated the integration made with the remaining ARCADIAN-IoT.

3.5.5 Future work

With respect to future developments, beyond the project lifetime, an essential aspect lies in augmenting the component with a policy management add-on. This envisioned extension would empower dynamic and seamless integration, modification, and removal of protection policies, aligning them precisely with the specific requirements of each deployment use case. This endeavor represents a key step towards enhancing the adaptability and responsiveness of the self-protection component. By affording a flexible and intuitive interface for policy administration, we anticipate a more agile and customized approach to safeguarding devices, thereby fortifying their overall security. This expansion will further refine the efficacy and versatility of the system in meeting the evolving demands of diverse operational environments.

3.6 Network Self-healing

3.6.1 Overview

3.6.1.1 Description

Within the scope of the ARCADIAN-IoT project, UWS has developed the Network Self-healing component. This component is designed to mitigate the potential impact of a cyber-attack when protection rules for that kind of cyber-attacks are not installed in the system (e.g., Firewall rules not installed) and thus the attack has the potential to penetrate the concerned IoT infrastructure.

This NSH component is asserting what action should be taken and where. The actions can be diverse, such as performing a drop, redirecting traffic, or mirroring the flow. On the other hand, the location on where should be enforced is determined following logic predefined by the programmer, such as "near the source", "near destination", "n hops from the source", etc. These actions can be automated by the administrator through a policy engine in order to define different strategies for each type of attack. This decision is taken by the analysis of the alert triggered by the previous component in the loop, the Network Flow Monitoring.

Thus, the Network Self-Healing component is responsible for generating a set of healing instructions. These instructions are the extension of the information provided by the Network Flow Monitoring, in order to achieve a set of implementable actions into the real system with the intention to complement any missing information not indicated in the Alert with aspect such as: "default duration", "default way to perform the action", etc. These set of steps define precisely what action to take, where to take it, how to take it, how long to keep it active and when to take it, among others. Furthermore, this component oversees the computing of this "close to source" location where the attack must be stopped. This is made possible by the information that the UWS Resource Inventory Agent (RIA) subcomponent is periodically reporting about the 5G network infrastructure and topological information.

3.6.1.2 Requirements

A recall of the high-level requirement that has been previously defined and provided in deliverable D2.4 [2] is given below.

Requirement 3.6.1 – Distributed Healing: The Network Self-healing component will be able to perform protection/healing rules in a distributed way according to the topological information gathered from the infrastructure with the main intention to heal/protect the network against DDoS attacks.

3.6.1.3 Objectives and KPIs

All these KPIs are related to the main project objective "*Self and coordinated healing with reduced human intervention*" (as recalled in the Introduction).

KPI scope	
Provide self-healing for overlay networks and IoT networks supporting >= 4 encapsulation and tunnelling protocols widely used in overlay networks inherent to 4G/5G IoT mobile infrastructures such as VXLAN, GRE, GENEVE or GTP.	
Measurable Indicator	
The number of supported encapsulations and tunnelling protocols widely used in overlay networks inherent to 4G/5G IoT mobile infrastructures.	
Target value	***Achieved value***
>= 4 (VXLAN, GRE, GENEVE, GTP)	= 4 (VXLAN, GRE, GENEVE, GTP)

KPI scope	
Provide dynamic and distributed enforcing of protection/healing policies in 7 or more network segments (Edge, Core, RAN, Transport, Backbone, Enterprise, Backhaul, Midhaul).	
Measurable Indicator	
Number of network segments supported to get the metadata information.	
Baseline value	
Core network segment (1)	
Target value	***Achieved value***
>= 7 (Edge, Core, RAN, Transport, Backbone, Enterprise, Backhaul, Midhaul)	= 7 (Edge, Core, RAN, Transport, Backbone, Enterprise, Backhaul, Midhaul)

KPI scope	
Provide network topology understanding to reduce human intervention (towards 0% of human intervention) supporting the coordination of recovery to pre-defined trust levels (>= 95% of flow services prior to anomalous behaviour) using such topology.	
Measurable Indicator	
% Recovery of the flow services prior to anomalous behaviour. Reduce human intervention to the strictly required, in healing and recovery procedures.	
Baseline number	
0% service recovery with 0% human intervention 100% service recovery with 50% human intervention	
Target value	***Achieved value***
> 95% service recovery Towards 0 % human intervention	= 97.5% service recovery Towards 0 % human intervention

3.6.2 Technology research

3.6.2.1 Background

The research carried out by UWS within ARCADIAN-IoT project has unveiled that the closest state of the art associated to the Network Self-healing component is the one related to Intrusion Prevention Systems (IPSs). These systems usually perform autonomous inspection of traffic and enforce the rules to heal and protect the network if there is a cyber-attack. The main problems with these tools are:

- They are traditionally designed for pure IP networks and thus they do not provide support for overlay networks nor for IoT network protection.
- They are usually installed in a component, which is deployed in the middle of the data path, and this is the only security control point available in the infrastructure and thus they do not provide support for dynamic distributed enforcing of protection/healing policies.
- They do not understand the network topology and thus are not able to take plans based on such topologies, especially the topology of IoT networks.

3.6.2.2 Research findings and achievements

We have designed and developed the Network Self-healing component based on the previously identified problems. The Network Self-healing component provides Prescriptive Analytics using the data received in real time to determine which actions should be enforced in order to mitigate an alert. Thus, when an attack has been detected we need to know:

- WHAT action should be taken
- WHERE this action should be enforced
- WHEN it must be enforced
- For HOW LONG it must be active.

The architecture of the Network Self-healing component enables the integration within a loop together with Network Flow Monitoring and Network Self-protection components. This loop provides the detection of alerts in overlay and IoT networks. These alerts are detected in distributed instances of Network Flow Monitoring deployed along the IoT multi-tenant infrastructure. Thus, the Network Self-healing component is in charge of receiving this information in a centralized instance. The information shared from the Network Flow Monitoring instances include (i) information about the malicious flow, (ii) information about the rule that has raised the alert, and (iii) the point where it has been detected (see Section 3.1.2.3).

The Network Self-healing component is composed of two subcomponents: the UWS Self-Healing Decision Maker (SHDM) and the UWS Resource Inventory Agent (RIA). RIA has been designed and a functional prototype has been implemented. It is a distributed component that must be deployed at least in the same machines of the data plane where the Network Flow Monitoring and the Network Self-protection components instances are deployed. RIA is in charge of the discovery of the network topology, so that SHDM can identify the point where the alert has been raised and where it has to mitigate the attack. RIA uses the Linux Inventory Tools (Figure 41) such as *lldpcli*, *ispci*, *iproute2*, *lshw*, *brctl or ovs-vsctl* juts to mention a few to collect the topological information in the machine where it is installed. This information is reported periodically to an internal topology interface where SHDM is listening.

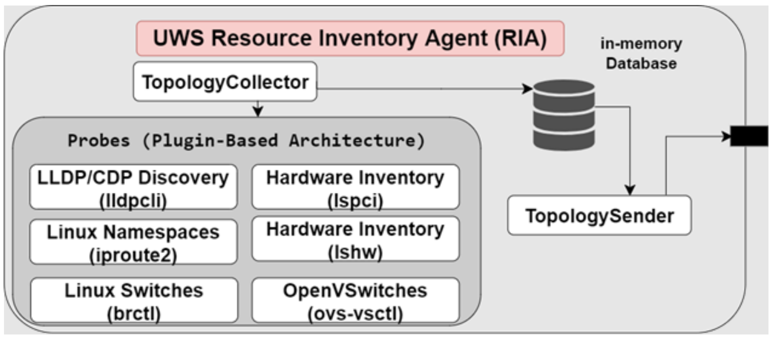

UWS Resource Inventory Agent (RIA)
in-memory Database
TopologyCollector
Probes (Plugin-Based Architecture)
LLDP/CDP Discovery (lldpcli)
Hardware Inventory (lspci)
Linux Namespaces (iproute2)
Hardware Inventory (lshw)
Linux Switches (brctl)
OpenVSwitches (ovs-vsctl)
TopologySender

Figure 41 - Architecture design of the UWS Resource Inventory Agent

SHDM has been designed and prototyped so that it will be a centralised component that implements the main functions of the Network Self-healing component. It receives the alerts from the Flow Monitoring component and the topological information from RIA. SHDM uses a set of Healing Decision Rules to provide Prescriptive Analytics and autonomously decide the WHAT, WHERE, WHEN and HOW LONG parameters previously cited. This information is sent as a Network Healing Instruction to be collected from the exact instance of the Self-protection component determined by the WHERE parameter.

3.6.2.3 Produced resources

Due to the application of IPR safeguarding measures, UWS has no intention to make any release of the source code for this component and thus it will not be made available in any public or private repository or server outside of our premises. For integration purposes, a functional prototype of the Network Self-Healing component will be released for the ARCADIAN-IoT consortium.

3.6.3 Design specification

3.6.3.1 Logical architecture view

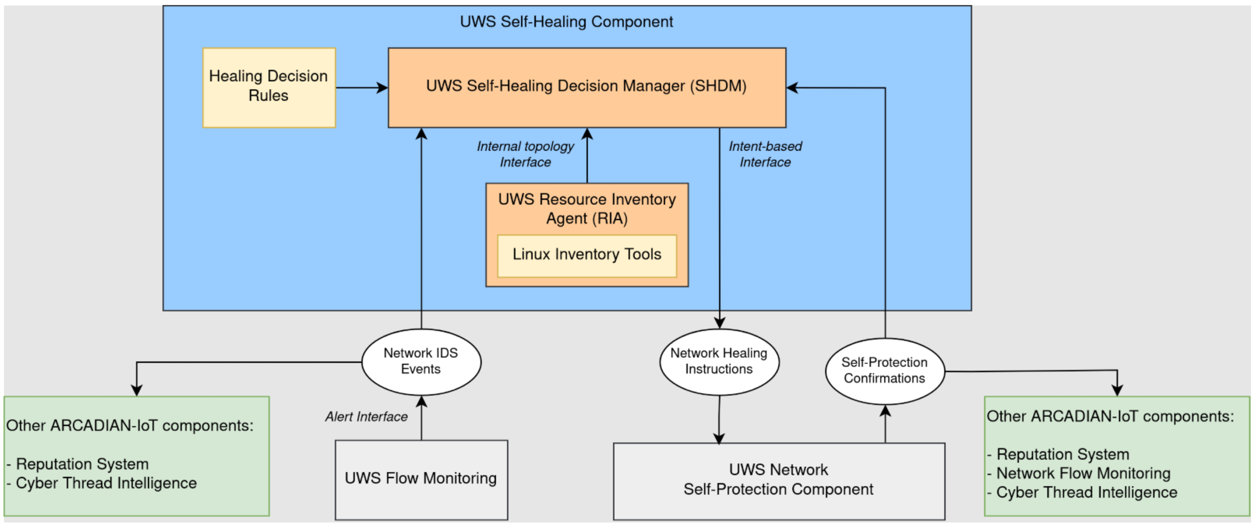

UWS Self-Healing Component
Healing Decision Rules
UWS Self-Healing Decision Manager (SHDM)
Internal topology Interface
Intent-based Interface
UWS Resource Inventory Agent (RIA)
Linux Inventory Tools
Network IDS Events
Network Healing Instructions
Self-Protection Confirmations
Alert Interface
Other ARCADIAN-IoT components:
- Reputation System
- Cyber Thread Intelligence
UWS Flow Monitoring
UWS Network Self-Protection Component
Other ARCADIAN-IoT components:
- Reputation System
- Network Flow Monitoring
- Cyber Thread Intelligence

Figure 42 - Architecture of the Network Self-healing component.

The Network Self-healing component in ARCANDIAN-IoT is based on an autonomous loop, where the following components and subcomponents are involved (see Figure 42):

- First, the ***Network Flow Monitoring*** component (Section 3.1) provides the sensing and detection capabilities of a cyber-attack such as DDoS. It allows to detect the cyber-attack.
- Second, the subcomponent ***Resource Inventory Agent (RIA)*** performs the periodical reporting of the IoT network infrastructure with the intention to allow an effective self-healing decision-making process using the topological information.
- Third, the subcomponent ***Self-Healing Decision Manager (SHDM)*** is in charge of determining what is the best plan to heal the network against that type of cyber-attack, including advanced intelligence aspects related to the device on where to stop the attack, the interface inside of such device and the sense of the communication flow passing for such interface, to inform about what is the best effective way to perform the healing of the network and also to send such information to the associated self-protection component;
- Finally, the fourth component is the ***Network Self-protection*** component (Section 3.4) that executes the mitigation of the attack, and finally deploying the necessary countermeasures to enforce the mitigation actions (e.g., traffic blocking/dropping, traffic mirroring, etc.).

3.6.3.2 Sequence diagrams

Figure 43 shows the sequence diagram of the Network Self-Healing component including the interactions with Network Flow Monitoring (NFM) and Network Self-Protection (NSP) components. It includes both message exchanges Network IDS Events and Network Healing

Instructions to show how they interact with each other. The loop on top refers to the periodic reporting of the topology from the -Healing Decision Manager (SHDM). This loop happens in all the distributed instances of RIA at the same time. The interaction below shows how NFM publishes the detected alert to the Network IDS Events exchanged the SHDM consumes that message and performs the prescriptive analytics. When SHDM has calculated the healing instruction, this is published to the Network Healing Instructions exchange. The published message will contain information so that only the specified instances of the NSP component receive such instruction.

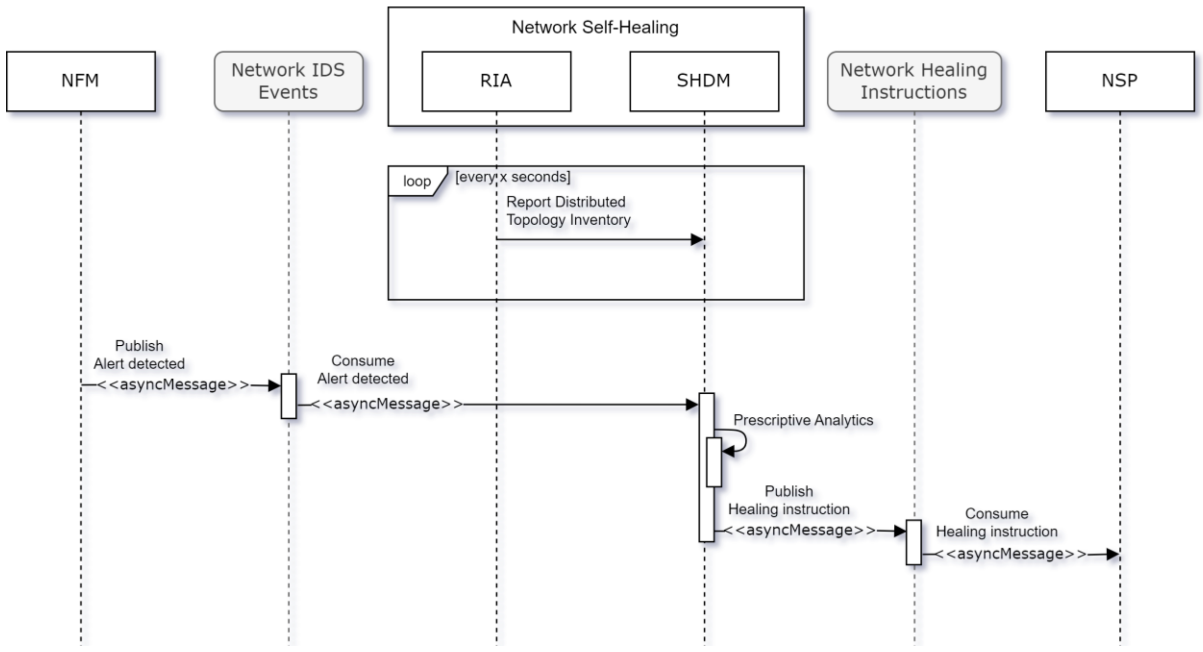


Figure 43 - Sequence diagram of the Network Self-healing component

3.6.3.3 Interface description

- Network IDS Events is the interface where the Alerts from the Network Flow Monitoring are published. All details and fields of these Alerts are described in previous Section 3.1.2.3.
- Network Healing Instructions is the interface that the NSH component uses to provide the Network Self Protection component the necessary information about what, where, when and for how long to mitigate the threat happening in the detected network segment.
- Network Self-Protection confirmations is the interface that the Network Self-Protection is using to provide information about the successful enforcement of the security rule provided by the Network Self-Healing component.

Partner	Component	Channel	Exchange
UWS	Network Flow Monitoring	RabbitMQ AMQP 0.9.1	Network IDS Events
UWS	Network Self-Protection	RabbitMQ AMQP 0.9.1	Network Healing Instructions
UWS	Network Self-Protection	RabbitMQ AMQP 0.9.1	Network Self-Protection Confirmations

Table 18 - Network Self-healing External Interfaces

3.6.3.4 Technical solution

3.6.3.4.1 Deployment architecture view

Figure 44 denotes the global deployment architecture view followed by the UWS cognitive loop for network security components alongside an entire 5G network infrastructure. The 5G architecture is composed of different network segments and layers. It can be differentiated from the Radio Access Network (RAN) segment, where IoT devices are connecting to the different 5G gNB with its respective Distributed Unit (DU) that connects to the consequent network segment, the Edge. This network segment oversees the responsibility of deploying and managing the digital services from the rest of the stakeholders involved in the system, as Digital Communication Services Providers. They use their virtual Central Units to manage communications across the rest of the infrastructure. In addition, in the Edge segment can be found Multi-access Edge Computing applications in order to approach computing services and capabilities to the end user. The transport network segment is the one between the Edge and Core premises, where the traffic is routed from the specific geographically distributed Edge to the centralized Core. The last network segment in the infrastructure is the Core, where first user authentication takes place. Once the communications leave the Core segment, they are forwarded to an inter-domain network segment outside the infrastructure provider's domain.

The Network Self-Healing component is instantiated in a management layer, which will develop the different communications with the rest of the cognitive protection network security loop. However, only the NSH Decision Manager is centralized deployed. The Resource Inventory Agents are distributed alongside the network infrastructure, recording and publishing fine-grain topology metadata that the NSH Decision Manager needs to perform its task. Once an Alert from the (centralized or distributed) Network Flow Monitoring component is received to the NSH Decision Manager, it will perform the prescriptive analysis that will ensure the best mitigation location and action to be enforced.

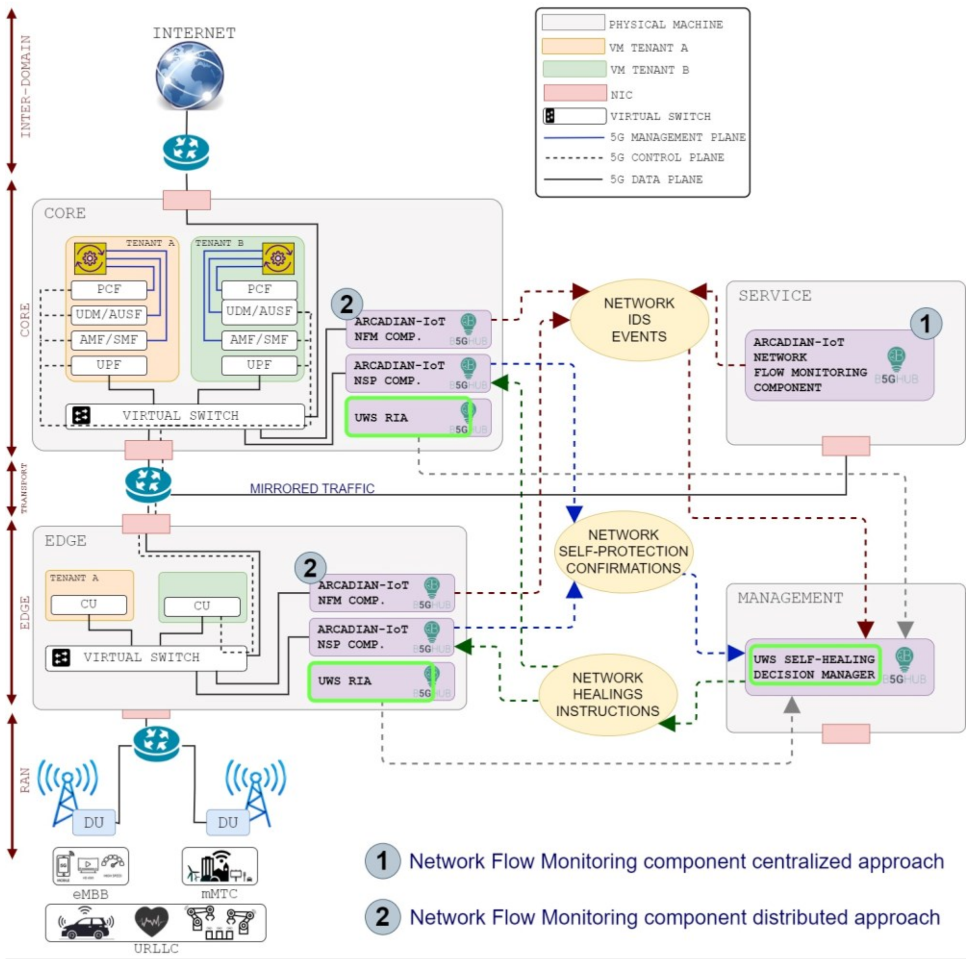


Figure 44 - Deployment architecture view of the Network Self-healing component

3.6.3.4.2 *API specification*

The Network Self-Healing component is using a message bus publication/subscription channel to interact with the rest of the components in the ARCADIAN-IoT framework. This component uses three different exchanges: the first is the Network IDS Events, where the Network Self-Healing will be retrieving Alerts. As a second exchange, the NSH component will be subscribed to the Network Self-Protection confirmations. This information will be used to confirm the effectiveness of the cognitive self-protection security network loop. Finally, the NSH will be subscribed to the Network Self-Protection confirmations. This information will be used to confirm the security rule has been already enforced by the NSP component and thus the effectiveness of the cognitive self-protection security network loop will be also confirmed.

3.6.4 Evaluation and results

The Network Self-Protection ARCADIAN–IoT component was validated in the same testbed described in section 3.1.4 for the Network Flow Monitoring (NFM). This testbed integrates the 3 architectural components conforming the Network Self-Healing Loop, that is, the NFM, the Network Self-Healing and the Network Self-Protection. Thus, the testbed configuration and different scenarios shown in Table 4 are the same for the 3 components. Section 3.1.4 provides a complete description and discussion of the testbed.

A notable observation can be made by comparing the behaviour of the NSH component in both

scenarios A and B described in section 3.1.4. A first glance at Figure 20 in section 3.1.4, It becomes evident that the execution time of NSH increases linearly as the number of IoT devices grows. However, it's important to note that in scenario A, the execution time is consistently higher across all settings compared to scenario B. For the most demanding scenario involving 64 IoT devices, NSH needed 33.8 seconds in scenario A, whereas it took 19.4 seconds in scenario B. As describer in section 3.1.4, the network topology in scenario A comprises 2 Edge Compute nodes and 2 gNBs, each one linked to a Data Service Provider (DSP), resulting in 4 action points for enforcing attack mitigation. The Self-healing Decision Maker (SHDM), serving as the centralized and cognitive component of the Cognitive Self-protection framework, is tasked with identifying the optimal location for threat mitigation, emphasising proximity to the source. As such, all the computational workload for determining the optimal mitigation location in scenario A falls upon the SHDM. However, in scenario B, not only does the number of action points for mitigation increase, but also the number of distributed Real-time Information Acquisition (RIA) components responsible for real-time network topology information dissemination. This is a crucial reason why NSH exhibits improved performance in scenario B, underscoring the advantages of workload distribution when confronted with a consistent number of attackers.
Furthermore, it is essential to emphasize that all the configurations outlined in Table 4 encompass scenarios and topologies characterized by the involvement of multiple stakeholders, the incorporation of diverse network segments, and the utilization of various overlay network layers within the network architecture. Consequently, the KPIs for the NSH component have been comprehensively addressed and successfully attained across these experimental scenarios.

3.6.5 Future work

Beyond ARCADIAN-IoT - Extending NSH Capabilities to Address New Threats: A promising avenue for future research involves the expansion of Network Self-healing (NSH) capabilities to effectively counteract emerging network-based cyberthreats. This entails exploring novel threat scenarios and considering various actuation points within the system. Rather than exclusively focusing on scenarios involving Open vSwitch (OVS) and a "close to source" policy, researchers should consider diverse use cases that require distinct decision-making and analysis. For instance, investigating scenarios where segments of the telecommunications infrastructure have been compromised and are engaging in spoofing activities, particularly regarding Digital Service Providers' information, would contribute to a more robust and adaptive NSH framework.

4 COMMON PLANE

4.1 Hardened Encryption with SIM as Root of Trust

4.1.1 Overview

4.1.1.1 Description

ARCADIAN-IoT aims at providing cryptographic mechanisms to secure and authenticate private IoT data. A basic approach is to employ symmetric encryption protocols allowing devices, servers, and users to secure their data for transmission, storage, or other purposes, and signature protocols allowing the entities to authenticate the data. Unfortunately, this approach implies that a big amount of cryptographic material, such as secure keys, needs to be managed. On one hand, keys could be leaked or breached, especially if a single authority server has access and stores all of them. Such an event could result in a leakage of data, injection of fake data, impersonation, or other security attacks. On the other hand, such a system does not offer much flexibility, since data can only be encrypted for one receiver, where again the encryptor needs to trust in an authority server, that the key management was not compromised.

ARCADIAN-IoT's Hardened Encryption (HE) component aims to overcome these limitations by providing a system that is more flexible, decentralized, minimizes the trust in central components, and is further hardened with information provided by a well-accepted hardware-based Root of Trust (RoT) – the SIM (Subscriber Identity Module) present in all cellular IoT and consumer devices.

In this section we provide the description of the encryption system based on the (decentralized) Attribute Based Encryption (ABE) paradigm hardened by a SIM hardware-based Root of Trust (RoT). The system can be applied to settings with devices having at least a basic processing unit able to perform encryption/decryption operations themselves, such as use cases in ARCADIAN-IoT WP5 Domains A, where the devices to manage are phones/tablets and drones. Note that in Section 4.2 another subcomponent of WP3 Task 3.2 is described named Hardened Encryption with cryptochip system, which covers the scenario in which the IoT devices are computationally more limited and hence embedded crypto chips or independent / external crypto chip modules were developed.

The HE with SIM as RoT consists of three subcomponents. The first one is a **software library** that devices, servers, and other entities can use to encrypt or decrypt/access the data. The main paradigm we build on is so-called Functional Encryption (FE), in particular, the Attribute Based Encryption (ABE) subfield. ABE allows participants to secure their data based on policies that determine who can access their data. This introduces a flexibility into the system giving the choice on who can decrypt the data in the hands of the encryptors, on one hand, while on the other hand, eliminates the need of an external central authority managing the access rights. Moreover, the policies that can be specified can range from simple ones, such as specifying exactly for which devices the data is encrypted, to more complex in which groups of users can be specified as decryptors, without increasing the complexity of the encryption or sizes of ciphertexts. This also drastically reduces the number of keys needed in the system.

Secondly, it includes a **decentralized key management system** for distributing the (ABE) keys and access rights among the entities in the system. This eliminates a single point of failure, such as a single server having access to all the cryptographic material. The system is easy to manage and can be deployed in multiple flavours: with multiple authorities handling separate keys, as a Multi-Party Computation system, or even as a centralized service if this is desired. The key management system is integrated with the blockchain solution provided in ARCADIAN-IoT framework, with the framework's Authentication component based on Self-Sovereign Identity and hardware-based RoT, as well as with the Reputation System.

Thirdly, to increase the trust in the system, prevent impersonation attacks, data poisoning and similar breaches, the digital signing of encrypted payloads will be enabled by **SIM hardware-based RoT**. Such signatures will guarantee the authenticity of data without the risk of cryptographic keys being exposed, due to being generated and embedded into the hardware secure element itself. Considering the large number of cellular devices (with any SIM form for connecting to a cellular network), using the SIM as hardware RoT has the advantage of not having to add new hardware secure element for this security feature in most IoT devices, ensuring a potential high impact of the solution. The SIM technologies developed build on GSMA's IoT SAFE specification[25] and are ready for any SIM form, including the latest ones: SIM, eSIM, iSIM.

4.1.1.2 Requirements

A recall of the high-level requirements that have been previously defined and provided in deliverable D2.4 [2] is given below.

Requirement 4.1.1 – Encryption mechanism: Enable secure and lightweight encryption with access policy strengthened with hardware-based RoT information.

Requirement 4.1.2 – Secure key-generation: Provide secure and scalable key management and delegation synchronized with the decentralized identity management.

4.1.1.3 Objectives and KPIs

With the aim of fulfilling the project's objectives, the Hardened Encryption component will be evaluated against the following KPIs that are related to the main project objective "Provide a Hardened Encryption with recovery ability" (as recalled in the introduction).

KPI scope	
Provide an encryption library enabling fine-grained access control over data with APIs in at least 4 programming languages and demonstrate its use on at least 2 types of devices/platforms.	
Measurable Indicator	
Number of platforms and programming languages supported.	
Target value	***Achieved value***
4 programming languages, 2 types of devices/platforms.	Supporting Python, Java, Go, JS and C tested on Android phones, Raspberry Pi, servers, embedded in web browser.

KPI scope	
Add Root of Trust information to encrypted data with SIM based signatures.	
Measurable Indicator	
1. Use of SIM as RoT in hardened encryption processes (Y/N) 2. SIM time to sign a payload (SHA256) 3. Number of different devices where the innovation is demonstrated	
Target value	***Achieved value***
1. Y	1. Y

[25] https://www.gsma.com/iot/iot-safe/

2. RoT signature time: < 2 seconds 3. 2	2. RoT signature time: 0.5 seconds 3. 2: SIM security applet is agnostic to the device where it runs; Android and Python middleware libraries provided for demonstration in 2 (according to IoT solution providers requests); for Android, an authorization SIM applet had to be delivered

KPI scope	
Provide a secure and scalable key management and delegation synchronized with the decentralized identity management, multi-factor authentication and self-recovery.	
Measurable Indicator	
Amount of integrations with other ARCADIAN-IoT components	
Target value	***Achieved value***
Decentralized key management integrated with other ARCADIAN-IoT components. 5 Components	Decentralized key management deployed with multiple key authorities or as a Multi-Party Computation service and integrated with other ARCADIAN-IoT Authentication, Permissioned blockchain, Self-Recovery and Reputation system. 7 completed integrations, with 3 still ongoing at the time.

KPI scope	
Provide efficient implementation of the process with encryption times comparable to the state-of-the-art results on multiple devices, and with RoT signatures in acceptable time for the communication processes.	
Measurable Indicator	
The encryption, decryption (dependent on the access policy complexity) and RoT signature time complexity.	
Benchmarking	
The benchmarks reported in [15] with a Java implementation that serve as a reference, although a different ABE scheme was used: Encryption: • A laptop with 1.60GHz Intel Quad-Core i7: ~160ms for a policy with 5 attributes • An Android phone with 1.60GHz Intel Atom Z2460: ~2.5s for a policy with 5 attributes Decryption: • A laptop with 1.60GHz Intel Quad-Core i7: ~250ms for a policy with 5 attributes • An Android phone with 1.60GHz Intel Atom Z2460: ~6s for a policy with 5 attributes	
Target value	***Achieved value***
Comparable to the state-of-the-art references: - Under 100ms on laptops and phones for encryption and decryption of ciphertext with a policy having 5	On a laptop with 1.70GHz AMD Ryzen 7 PRO 4750U with Go based implementation and Python bindings for MAABE and FAME schemes, respectively: • Encryption 14ms and 32ms for a

attributes. - Under 5s for limited device such as Raspberry Pi for encryption and decryption of ciphertext with a policy having 5 attributes.	policy with 5 attributes, respectively. • Decryption 10ms and 7ms for a policy with 5 attributes, respectively. On an Android phone Samsung A33 with 2.4 GHz Cortex-A78 with Go based implementation and Java binding: • Encryption 26ms and 52ms for a policy with 5 attributes, respectively. • Decryption 23ms and 20ms for a policy with 5 attributes, respectively. On a Raspberry Pi 3 Model B+ with 1.4GHz Cortex-A53 with Go based implementation and Python binding: • Encryption 4.7s and 12s for a policy with 5 attributes, respectively. • Decryption 3.1s and 12s for a policy with 5 attributes, respectivley.

4.1.2 Technology research

4.1.2.1 Background

4.1.2.1.1 Attribute-Based Encryption (ABE)

Attribute-Based Encryption (ABE) represents a family of encryption schemes first introduced in 2005 by A. Sahai and B. Waters [16], that mathematically ingrains access control (specifically, access policies) directly into the encryption process itself. The family is commonly divided into two distinct subfamilies of schemes: Key-Policy ABE and Ciphertext-Policy ABE (CP-ABE) (see [17]]). As the former is less suitable for use in this project, we focus on the latter. The principal idea of CP-ABE is that the ciphertext is encrypted with an access policy defining which attributes an entity needs to possess in order to decrypt the ciphertext. The attributes here can be arbitrarily defined by a use-case, ranging from single entities (e.g., specifying the names of the devices that are allowed to decrypt) to groups all having the same rights (e.g., all devices in the same group, such as a group of phones of doctors or nurses, have the same attribute). The access policy is normally in the form of a Boolean formula, i.e., the necessary attributes connected by the AND and OR logical connectives. The exemption of the NOT logical connective makes it impossible to exclude entities possessing more attributes than necessary, which is in line with the idea that entities may choose to hide parts of their identity in the Self-Sovereign Identity model. Entities wishing to decrypt the data encrypted with them in mind have to obtain the attribute/decryption keys (one per attribute) from the Attribute Authority that checks the entities' eligibility for attribute keys and delegates the keys to the respective entities. The data is then decrypted if and only if the set of attribute keys belonging to the entity satisfies the access policy. This procedure has many advantages:

- The users need to maintain only their set of attribute keys to be able to decrypt all the data encrypted with them in mind, since data is always encrypted with the same public key (technically belonging to the Attribute Authority), instead of managing a complicated web of symmetric and asymmetric encryption keys.
- The data access is specified by an encrypting device; hence no trust is needed in a central component to handle access to data or even be directly trusted with unencrypted data. This eliminates a major point of failure.
- The functionality does not require a major computational bottleneck, since it can be implemented efficiently (comparable to standard public key schemes).

A notable problem with ABE is the existence of a single Attribute Authority. The Attribute Authority needs to be a trusted entity, since it possesses the master key (corresponding to the public key used for encryption) that is used to delegate decryption keys to all other entities in the network. In particular, it allows to derive decryption keys that can decrypt all data encrypted. Hence it represents a single point of failure and is an obvious attack target for adversaries and malicious actors who wish to decrypt sensitive data, enrol their own devices into the network, or disturb the network by preventing the distribution of new decryption keys. Fortunately, effort to decentralize this component have been made in the literature, see [18] for a survey of such approaches. We implement two decentralization approaches, namely we use an ABE scheme that allows to deploy multiple key authorities as well as we implement a Multi-Party Computation protocol that has the same functionality as a centralized authority but is secured against malicious corruptions and hence decentralizes trust.

In the recent years, various projects started implementing ABE and a more general Functional Encryption (FE) to practical scenarios. We use GoFE library developed in European H2020 project FENTEC as our base on which we implement the functionalities in this project.

4.1.2.1.2 SIM as Root of Trust (RoT)

The main background for the use of SIM as RoT in the Hardened Encryption process is GSMA IoT SAFE[26]. This industry fora working group and the technical specification it develops are still in an early stage and open for contributions. IoT SAFE aims to enable IoT device manufacturers and IoT solution providers to leverage the SIM as a robust, scalable, and standardized hardware RoT to protect IoT data communications. Some of its services are allowing IoT devices to perform mutual (D)TLS authentication to a server using asymmetric or symmetric security schemes; to compute shared secrets and keep long-term keys secret; and to allow to perform credential provisioning and lifecycle management from a remote IoT security service. IoT SAFE is compatible with any SIM form (SIM, eSIM, iSIM, …), uses the SIM as a ‘crypto-safe’ inside the device, provides/specifies a common API for the SIM to be used as a hardware RoT, and facilitates the provisioning of millions of IoT devices in what regards security schemes.

However, GSMA IoT SAFE focuses just on IoT devices, disregarding consumer devices like smartphones, which may represent persons activity in digital systems, namely in IoT solutions. Considering that ARCADIAN-IoT also focuses on persons and their activity as part of the IoT ecosystems, and has use cases joining exactly persons, IoT devices, Cloud services and others, it is relevant to consider the use of SIM as RoT in consumer devices as well. In this sense, there is significant background knowledge to consider.

Focusing on Android devices, the type of smartphones we will consider for prototyping in this project, the operating system has concrete authorization rules for mobile applications’ communication with the SIM. In Android’s technical documentation about carrier functionalities can be found the UICC Carrier Privileges[27] (UICC stands for Universal Integrated Circuit Card). The carrier can be understood as the SIM provider and manager, and UICC can be understood as the hardware secure element that has SIM profiles. Since Android 5.1 there is a mechanism to grant special privileges for APIs relevant to the owners of universal integrated circuit card (UICC) apps. The Android platform loads certificates stored on a UICC and grants permission to apps signed by these certificates to make calls to special APIs. Carriers have full control of the UICC, so this mechanism provides a secure and flexible way to manage apps from the mobile network operator (MNO or MVNO) hosted on generic app distribution channels (like Google Play store) while retaining special privileges on devices and without the need to sign apps with the per-device platform certificate or preinstall as a system app.

The rules on the UICC – rules that define which app can interact with the SIM – should be compatible with the GlobalPlatform Secure Element Access Control specification Figure 45.

[26] https://www.gsma.com/iot/iot-safe/

[27] https://source.android.com/docs/core/connect/uicc

These authorization rules include the app name (full package name string) and the app SHA-1 certificate with which the app is signed, and an 8-byte bit mask representing 64 separate permissions. The SHA-1 certificate is the only field that is mandatory.

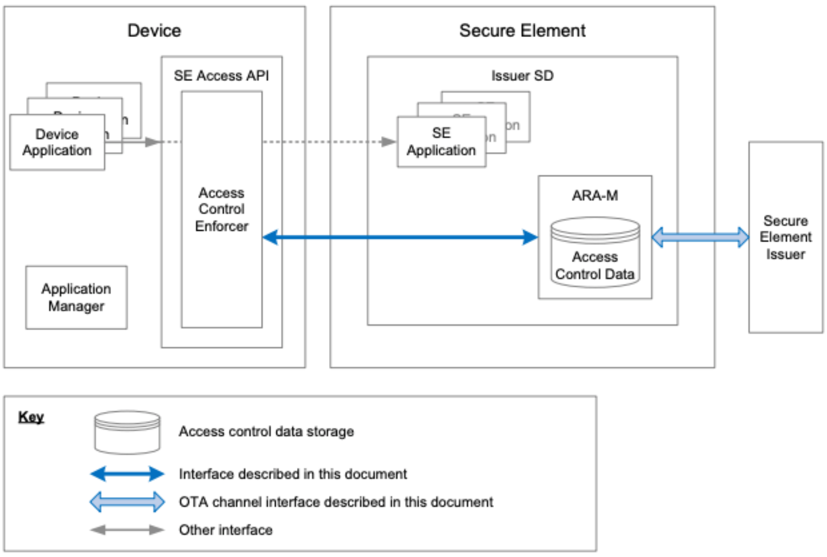


Figure 45 - Secure Element Access Control Architecture[28]

Figure 45 depicts GlobalPlatform's Secure Element Access Control Architecture. Interpreting the figure in the context of the Android Operating System (OS) and Android's Carrier Privileges previously mentioned, it's possible to start by understanding the Device Application as an Android Application (or app). For being possible for this app to interact with a SE (Secure Element) Application, the Secure Element Issuer, meaning the carrier in Android's documentation, needs to securely provision a rule identifying that app to the Access Control Data database, which needs to exist in an Access Rule Application Client (ARA-C) running on the Secure Element. This provisioning happens via standardized GSM over the air (OTA) services used for the secure communication between carriers and SIMs. On the boot up of the device, an Access Control Enforcer (e.g. from the Android OS) reads all the rules that identify which apps can access the Secure Element (the SIM in the context of this project). There is also a need of a middleware API to access the SE Application, depicted in this diagram as part of the Access Control Enforcer.

The research for using the SIM as RoT in Hardened Encryption reported in the next sections uses and departs from the concepts conveyed by GSMA IoT SAFE, Android Carrier Privileges and GlobalPlatform Secure Element Access Control.

4.1.2.2 Research findings and achievements

4.1.2.2.1 Library for Hardened Encryption with RoT Signatures

One of the main outcomes of the development of the Hardened Encryption component is a library enabling entities in the ARCADIAN-IoT framework to secure and authenticate their data. Due to the advantages of the ABE paradigm and the ability of SIM to anchor the RoT in hardware, these two technologies were chosen to harden the standard approaches in IoT (and mobile devices) for encrypting and authenticating the data.

In the joint studies made by 1GLOBAL and XLAB, several hypotheses/approaches were

[28] https://globalplatform.org/wp-content/uploads/2014/10/GPD_SE_Access_Control_v1.1.pdf

considered on how to join ABE and SIM. Examples are: (i) implementing the encryption mechanisms within the secure element itself; or (ii) establishing a secure communication channel for IoT devices. Not considering ARCADIAN-IoT context, both hypotheses were feasible: the SIM has cryptographic encryption capabilities and allows the establishment of secure channels (e.g., (D)TLS). However, considering the envisioned IoT solutions of the targeted domains, the first hypothesis was excluded due to the amount of data to encrypt (e.g., sets of photos from drones), which is unfeasible to be processed at the secure element. The establishment of a secure communication channel is a feasible approach but does not relate completely with the objective of the HE process of being an encryption process that is strengthened with RoT information. Considering the context and the objectives, the hypothesis selected for being prototyped is a novel combination of the SIM abilities with the ABE previously described and summarized in Figure 46. It consists of using SIM RoT capabilities for generating unique key material and signing data encrypted at the device with the ABE (or the hash of that data). In this scenario, the ABE encryption that allows a fine-grained access control over encrypted data is strengthened with the RoT digital signature, performed with unique material generated and kept at the hardware secure element, avoiding thus, e.g., impersonation attacks (malicious agents sending data – fake, corrupted or with other intention – in the name of other devices).

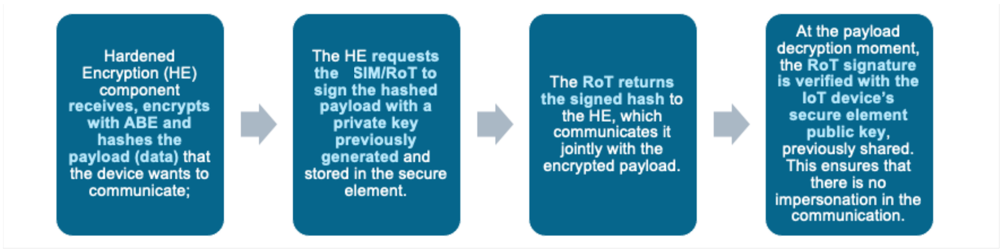

Hardened Encryption (HE) component receives, encrypts with ABE and hashes the payload (data) that the device wants to communicate;
The HE requests the SIM/RoT to sign the hashed payload with a private key previously generated and stored in the secure element.
The RoT returns the signed hash to the HE, which communicates it jointly with the encrypted payload.
At the payload decryption moment, the RoT signature is verified with the IoT device's secure element public key, previously shared. This ensures that there is no impersonation in the communication.

Figure 46 - Research hypothesis for the Hardened Encryption to join ABE with SIM as RoT

The library, to be deployed on devices producing and securing the data in ARCADIAN-IoT context, will thus consist of three main elements:

- Software implementation of ABE functionalities.
- SIM security applet running in a secure element of a device, enabling RoT signatures.
  - For Android devices a second SIM applet is mandatory, to receive authorization information related with the apps that can interact with the SIM.
- A middleware allowing to invoke the applet from the software part, compatible with the software running on the device.

The key generation in the hardware secure element can happen invoking, in the device, the middleware aforementioned, or via GSM standardized Over-The-Air (OTA) services. Upon the key generation request, the private key used for the RoT signatures is kept at the secure element and the public key that allows to validate the signatures is returned. Further details are provided in Section 4.1.3.

4.1.2.2.2 ABE software library for Hardened Encryption

An encryption library in an IoT setting needs to be implemented flexible enough to support multiple types of devices/platforms using multiple programming languages/frameworks. On the other hand, to implement algorithms enabling ABE functionalities, advanced cryptographic primitives need to be used, such as elliptic groups enabling bilinear pairings. One of the requirements of the HE component is that the encryption process is as lightweight as possible, hence not being a computational bottleneck.

To have an efficient implementation that can be used by all the entities in the system, we approached the problem by developing the library in programming language Go, due to its performance and capabilities to be cross-compiled for multiple platforms (phones, ARM based devices, etc.). In addition, we implement bindings to other programming languages, such as Pyhton, C, Java, etc., so that the functionalities can be used by many applications, without losing the efficiency of Go implementation. This extends the standard tests and applications of the ABE found in the literature [19, 20], where the academically developed ABE schemes were mostly tested on singular platforms. In fact, one of the KPI of HE component is to have efficient encryption process, comparable to the state-of-the art results, on multiple devices.

We have chosen to implement two ABE encryption mechanisms, that differ in the underlying cryptographic construction, as well as in the functionality they offer:

- (CP-)ABE scheme named FAME from [21]: A scheme designed to improve ABE efficiency that is based on the standard cryptographic assumption in pairing groups. It enables to encrypt data with specified policies using a single public key (that can be published at the start of the system). Policies can use arbitrary many attributes/roles, a feature also known as a large universe ABE. Originally, the scheme includes a single key management authority, that distributes private keys, which we decentralized using a Multi-Party Computation protocol, see Section 4.1.2.2.2.
- Multi-Authority (CP-)ABE scheme (MAABE) from [22]: A scheme that by design allows to initiate multiple key management authorities, that each distributes their own public keys and manage appropriate attributes/roles in the access system. A client encrypting can apply mixed attributes/roles managed by different authorities to the access policy, hence distribute trust. On the downside, applying complex access policies decreases the efficiency of the encryption/decryption. Moreover, all the attributes in the system must be known in advance, and the number of them impacts the size of the public keys.

Due to the efficiency, we do not use the ABE protocol directly on data, but rather in cooperation with standard symmetric encryption schemes, such as the AES block cypher in CBC mode or other. The idea is to use symmetric encryption with randomly generated keys, that are encrypted with ABE schemes and attached to encrypted data, see figure below.in a variety of scenarios, such as encrypting text, figures, streams of data, etc., with minimum computational expense.

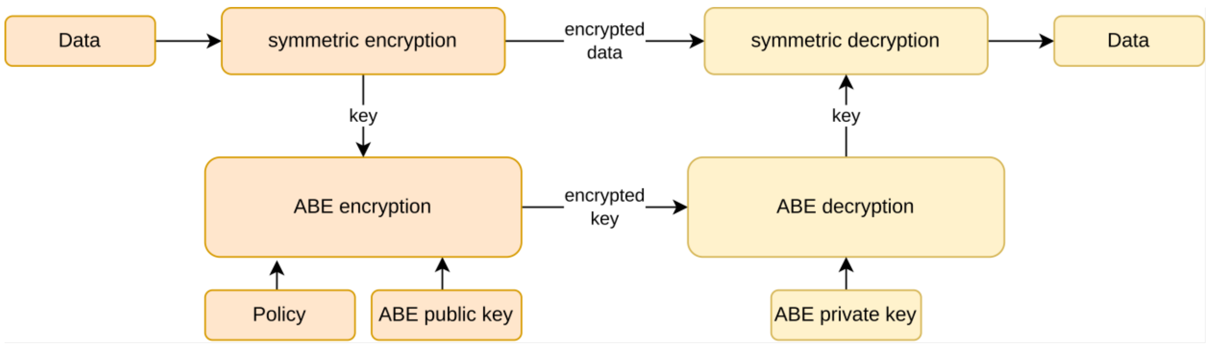


Figure 47 - The encryption and decryption process in HE

4.1.2.2.3 *SIM as hardware-based RoT for Hardened Encryption*

For using the SIM as hardware-based RoT in the Hardened Encryption process, the research led to the development of the several components previously introduced: (1) a SIM security applet able of running in any SIM form (SIM, eSIM, iSIM) that follows GSMA IoT SAFE specification; (2) 2 versions of a middleware for the encryption component to interact with the SIM security applet (one for Linux-based IoT devices and another for Android devices); and (3) an Access Rule Application Client (ARA-C) SIM applet, to settle the authorization rules for apps running on Android OS to interact with the SIM. Moreover, were developed all the OTA service scrips for the carrier to securely install, manage and interact with the new SIM applets above mentioned, according to GSM standards.

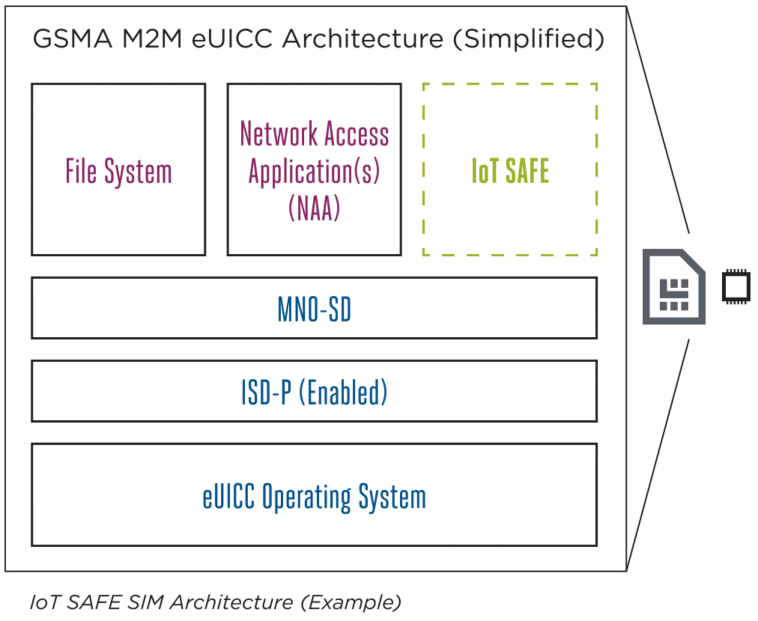

GSMA M2M eUICC Architecture (Simplified)
File System
Network Access Application(s) (NAA)
IoT SAFE
MNO-SD
ISD-P (Enabled)
eUICC Operating System
IoT SAFE SIM Architecture (Example)

Figure 48 - UICC general structure comprising IoT SAFE [29]

The SIM security applet leveraged GSMA IoT SAFE specifications and architecture, positioned within a UICC general structure as shown in Figure 48. Furthermore, this applet was also conceived to be able to receive, from the carrier, trust information related to the device where the SIM is. With this information the applet should perform self-protection and self-recovery actions (see Network-based Authorization component for more information). This component is part of a regular SIM profile for connectivity enablement, including the regular 1GLOBAL's MVNO (Mobile Virtual Network Operator) connectivity features for such. As previously mentioned, while the technology was built for running in any SIM factor (SIM, eSIM, iSIM), throughout the project was acknowledged that the best hardware for receiving this technology are Java Card 3.0.5 ready chips that run SIM Operating Systems capable of running the cryptographic functions specified in GSMA IoT SAFE (e.g. ST4SIM-200M, an eSIM from STMicroelectronics). Even though, the research also showed the possibility of using the technology in chip sets with older Java Card versions (if some compromises were made).

In what regards security, the physical characteristics of the embedded or integrated SIM forms (eSIM and iSIM) are, by design, more secure. These embedded or integrated SIM forms don't allow to use a specific secure element (with specific identity and key material) in a different device than the one it was intended to - the eSIM is soldered in the device board and the iSIM is part of the integrated circuit itself, both being non-removable.

As previously mentioned, the novelty of ARCADIAN-IoT SIM security applet starts by its compliance with GSMA IoT SAFE, which is expected to contribute to an alignment with industry standards. To these standards, are added the developments needed for the integration within ARCADIAN-IoT's Hardened Encryption as well as with SIM-based device Self-protection and Self-recovery. Furthermore, no use of IoT SAFE in consumer (Android) devices is found in the state of the art, and in ARCADIAN-IoT we also apply the SIM as RoT in these devices with wide-ranging market penetration. The intended scenarios for the ARCADIAN-IoT SIM security applet[30]

[29] https://www.gsma.com/iot/iot-safe/
[30] ARCADIAN-IoT SIM security applet is applicable and ready to run to any SIM form, having been thoroughly tested with several eSIM hardware (eUICCs) from several manufacturers

are the following:

- *M2M bootstrap scenario*: When a compliant IoT device is turned on for the first time it is provisioned with an ARCADIAN-IoT SIM profile, which includes the connectivity features and the security applet. Once the profile is activated in the device, it connects to 1GLOBAL network, which, by recognizing the device and the ARCADIAN-IoT SIM, requests, securely, over the air, the security applet to generate a public/private key pair. The public key is returned, provisioned to ARCADIAN-IoT key management services, and made available to the components that will need to validate the signatures in the payloads. Alternatively, the request for key generation in the secure element can be done by the Hardened Encryption component. For demonstration purposes, the second option was taken in the project.
- *Personal device bootstrap scenario*: When (or before) a user registers in an ARCADIAN-IoT app it is provisioned with an ARCADIAN-IoT SIM profile, which includes the connectivity features and the security applet. Once the profile is activated in the device it connects to 1GLOBAL network, which, by recognizing the device and the ARCADIAN-IoT SIM, requests, securely over the air, the security applet to generate a public/private key pair. The public key is returned, provisioned to the ARCADIAN-IoT key management services, and made available to the components that will need to validate the signatures in the payloads. Alternatively, the request for key generation in the secure element can be done by the Hardened Encryption component. For demonstration purposes, the second option was taken in the project.
- *Hardened Encryption process*: To ensure data privacy and security, all the sensitive outgoing data from a ARCADIAN-IoT IoT device, or app in a personal device, needs to be encrypted. The process, previously described, consists of firstly encrypting the data (ABE), which then requests the RoT to sign the encrypted payload (its hash), and finally sends it to the intended service (the encrypted data and the signed hash of the encrypted payload).
- *Signature validation*: ARCADIAN-IoT services that are intended to verify the data integrity (e.g., attestation component, or a decryption component) should have access to the public key that allows to verify the RoT signature of the payloads, being informed thus of its integrity.

These scenarios were the vision, iteratively enhanced throughout project, that guided the research.

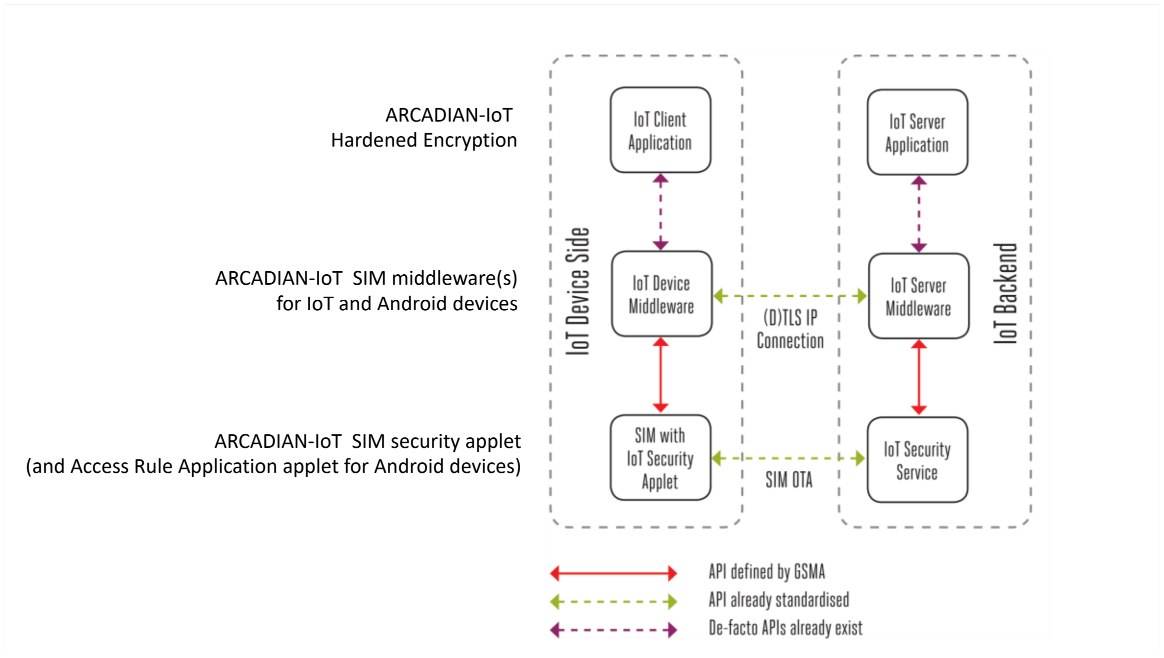


Figure 49 - ARCADIAN-IoT Hardened Encryption prototype mapped into GSMA IoT SAFE architecture[31]

Figure 49 depicts the match between ARCADIAN-IoT components with GSMA IoT SAFE components. The final prototypes include, besides the Encryption technology, 2 SIM middleware interfaces, for Linux-based IoT devices and Android devices, the SIM security applet with the functionalities described in this section, and an ARA (Access Rule Application) applet for the needed authorization processes for Android apps to be able to interact with the SIM.

#	Method	Brief description
1	**Generate Key Pair**	The **Generate Key Pair** method is used to generate an asymmetric key pair. Upon successful execution both public and private keys are updated in the applet key store with their respective value. The public key is returned to the caller, and the private key is kept in the secure element, without any method to request/access it.
2	**Compute Signature - Init**	The **Compute Signature – Init** command opens a session to compute a signature. The command can also be used to cancel a signature computation session.
3	**Compute Signature - Update**	The **Compute Signature – Update** command is used to provide the applet with reference data to compute a signature with the private key stored in the secure element
4	**Get Response**	The **Get Response** command is used to get the signed data (signed hash of the encrypted payload in ARCADIAN-IoT) from the SIM applet. This method is to be used after method #3.

Table 19 - Main methods from ARCADIAN-IoT SIM security applet

The methods in Table 19 are the key ones for fulfilling the intended functionality of the RoT in the HE process. These exist in the SIM security applet and are accessible from the middleware's developed, for IoT devices (Linux-based according to the IoT solutions requirements) and Android

[31] https://www.gsma.com/iot/iot-safe/

devices. For Android devices, as previously mentioned there is a second applet that defines the authorization of the applications that can interact with the SIM (ARA applet). These technologies were developed, deployed, and target of successful unit and functional testing, and integration testing within the HE process in 2 IoT solutions present in the project. Some results are presented in section 4.1.4.

4.1.2.2.4 Decentralized key management of ABE keys and SIM public keys

Every encryption or authentication protocol requires a key management system, enabling a distribution of private and public keys, and with that it determines the access policies and anchors the trust. In the case of the HE component, two types of keys need to manage:

- ABE keys: the public keys of the key generating authorities that are needed for devices to encrypt data with the desired policies as well as private decryption keys.
- SIM public keys: the signature verification keys of the devices participating in the platform, corresponding to private signing keys embedded in SIM.

The HE component is (due to the use of ABE and hardware-based RoT) designed to minimize assumptions on trustfulness of single devices and presents a step towards the zero-trust security. One of the identified threats in a naively designed system would be a key distribution with a central authority responsible to delegate (private) decryption keys and handle guaranties of public keys for encryption or signatures verification.

To mitigate this issue in ARCADIAN-IoT system, we have developed and implemented two approaches that enable to decentralize the trust in the cryptosystem.

- We use MAABE encryption scheme from [23] and implement a multiple key authority setup in order to mitigate a single point-of-failure weakness present in traditional ABE schemes. The scheme uses multiple Attribute Authorities that check the user's eligibility for decryption keys pertaining to presented attributes and then delegates the decryption keys to the user, as illustrated in Figure 50. The Attribute Authorities can serve decryption keys for a unique set of attributes, they can serve the decryption keys for the same set of attributes as other Attribute Authorities, or any combination of the two. The decryption policy can then be adjusted to require an almost arbitrary set of attributes from various Attribute Authorities for the user to possess in order to decrypt the data. The decryption policies can therefore be constructed in a way that it forces potential malicious actors to compromise many (or all) Attribute, not just a single one, in order to obtain a usable set of decryption keys, or in a way that maximizes redundancy to make the network resilient to DoS attacks. To the best of our knowledge, this is the first approach to use a multi-authority ABE scheme in the context of IoT devices.

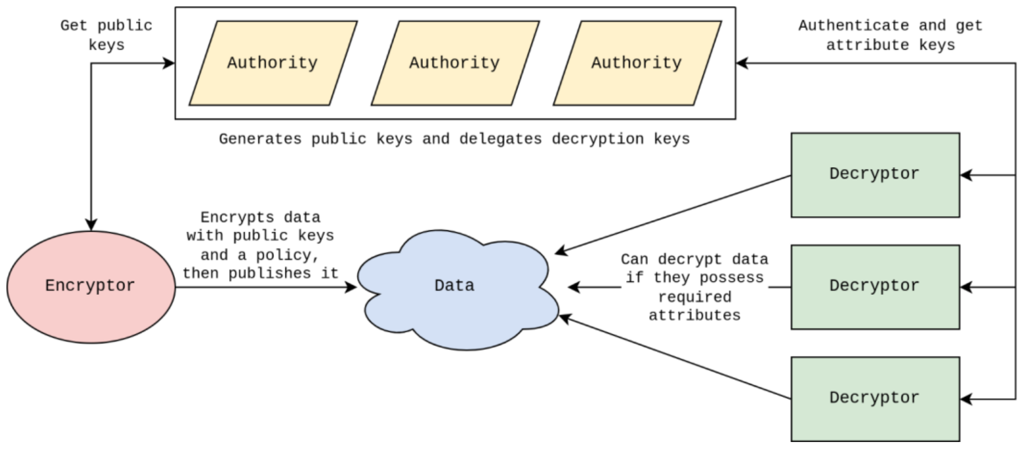

Get public keys
Authority
Authority
Authority
Authenticate and get attribute keys
Generates public keys and delegates decryption keys
Decryptor
Encrypts data with public keys and a policy, then publishes it
Encryptor
Data
Can decrypt data if they possess required attributes
Decryptor
Decryptor

Figure 50 - Multi-authority key management

- We use FAME encryption scheme from [24] and implement its key management in two ways. As a central key authority as originally designed or alternatively as a Multi-Party

Computation (MPC) system. In fact, we custom-designed an efficient MPC protocol based on well-established SPDZ MPC scheme [25]. SPDZ is an efficient, maliciously secure MPC protocol that can guarantee its security, as long as there is at least one MPC node in the system, that is not malicious. In our case, this means that we can deploy, instead of one single key management authority, a system of *n>1* computation nodes, that can securely communicate with each other and provide the same functionality. To compromise such a system and gain power over it, a malicious actor would need to compromise all the *n* nodes. This extends the trust in the system. The efficiency of SPDZ is based on the fact that the protocol can be split into 2 phases, usually named the offline and online phases. The offline phase is non-specific to the computation that needs to be done in the online phase; hence it can be done in advance. In the online phase (in our case, when a client requests a private key), the key management system computes the key for the specific ABE attributes. To have the online phase as efficient as possible, we set up the parameters (particularly certain prime numbers) of the SPDZ scheme to match the parameters of the ABE scheme. We implement the offline phase based on the Mauer scheme [26]. The maliciously secure protocol assumes an honest majority of the computation nodes.

With respect to the management of (SIM) public signature keys we take the following approach:

- We use the key management as a Certificate Authority (CA) that signs the public signature keys, assuming that the clients authenticate themselves (using ARCADIAN-IoT authentication).
- We publish the public keys on the ARCADIAN-IoT Permissioned blockchain ledger. This anchors additional trust in the public values.

Further actions were taken:

- We use the ARCADIAN-IoT blockchain ledger for publishing public ABE keys in a decentralized manner. This way we again eliminate the need for a single (for example certificate) authority, that needs to be trusted. Instead, we anchor the trust on a decentralized blockchain ledger
- We included in the architecture the integration of the HE component with the ARCADIAN-IoT (multi-factor) Authentication component. The authentication is needed in the onboarding process of devices in ARCADIAN-IoT framework, since the key management authorities need guaranties that they are distributing ABE decryption keys to the appropriate entities.
- We integrated the HE key management with the ARCADIAN-IoT reputation system. To be precise, the key management system gives information about the key requests to the Reputation system, that can use it to modify the trust score of a client in cases of malicious behaviour.
- Finally, the system of attributes used to specify encryption policies and distribute keys is designed to be aligned with the ARCADIAN-IoT identity. In particular, one of the types of attributes that can be used is the ARCADIAN-IoT unique name of each device or person in the system. Hence, devices will be able to specify access policies that are exactly determining the receiving entities.

4.1.2.3 Produced resources

In this section we report on all the outcomes of the HE component that has to be noted that will be in part restricted access but we open-sourced certain parts of the component.

The HE component includes multiple sub-repositories:

- **HE ABE library**: We have implemented the ABE schemes described in Section 4.1.2.2.1.Moreover, the library can be compiled for multiple devices. It was demonstrated on multiple devices including servers, phones, directly in a web-browser, and multiple models of Raspberry Pi. See Section 4.1.3.4 for the description of the interface to the library and Section 4.1.4 for the efficiency report. The library was developed with modern paradigm of continuous testing, tools for automatic detection of security issues, and includes unit and integration tests. Furthermore, it was tested in Domain A and C on various devices.
- The **SIM security applet** has also been developed and its implementation is independent of the device where the SIM will live and of the SIM factor. It currently has all the envisioned needed functionalities and throughout the project its performance (e.g. for performing digital signatures) was improved to a state that the partner IoT solution providers find to be good. The test results and its analysis can be found in section 4.1.4. An auxiliary applet (ARA applet, previously mentioned) is needed for authorizing mobile apps to interact with the SIM in Android ecosystems.
- The device middleware, for the HE communication with the SIM secure element also has been developed considering Linux-based IoT devices and Android devices. To allow for the integration in the different use cases, it is developed both in Python and Java. Both the SIM security applet and the device middleware were successfully tested, integrated in the HE setup and in IoT solutions from partners, with several different modems and SIM chipset manufacturers (which may influence the middleware development (AT) commands or the applet behaviour).
- **HE Key Management**: We have implemented in Go a complete key management system that manages ABE keys as well as the signature public keys and custom certificates, as described in Section 4.1.2.2.2. The system is easy to deploy since it is fully Dockerized. In particular, it can be deployed in 3 ways:
    - As a multi-authority service, i.e., handling MAABE keys.
    - As a decentralized service, i.e., as a Multi-Party Computation system handling ABE keys of the FAME scheme.
    - As a centralized service, i.e., as a trusted authority that handles FAME keys.

See Section 4.1.3.4 for the description of the interface.

- **HE Demos**: We provide a repository with demos of how to use the HE library, RoT signatures and HE key management with multiple APIs, including Python, Go, Java and JS. For example, a demo that uses Python API shows how to interact with a HE key manager, encrypt data with a specific ABE policy, sign the data with SIM based RoT, and decrypt the data if the policy allows it.
- **HE Android App Demo:** We provide a demonstration app showcasing how to encrypt/decrypt data inside an Android App and enhance the encrypted payloads with RoT signatures.

As part of the XLAB's dissemination activities, we also provide open-source code, that helps developers adapt and integrate ABE technologies into other projects. In particular, we made the following repository public: https://github.com/xlab-si/AHE It includes the HE component stripped of all the other ARCADIAN-IoT limited subcomponents. With it, developers can use the ABE libraries, deploy a key management, and integrate with their own components, such as (non-ARCADIAN-IoT) authentication. It includes demos to help the adoption.

4.1.3 Design specification

In this subsection we provide further design specifications. Firstly, we provide the architecture view explaining how the subcomponents of HE interact with each other. We enhance it with a sequence diagram, explaining how the interactions happen during the onboarding of a device, encryption, and decryption process. Finally, we provide information on how to use the HE component by providing the interface description of the HE library.

4.1.3.1 Logical architecture view

Architecture view of the HE with ABE encryption and SIM signatures is visualized in Figure 51.

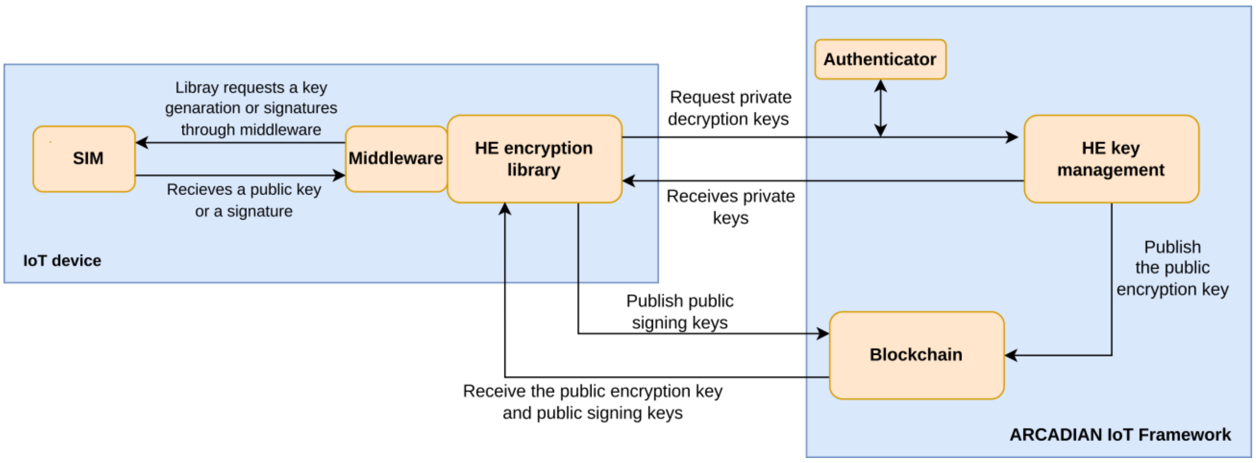

Authenticator
Libray requests a key genaration or signatures through middleware
Request private decryption keys
SIM
Middleware
HE encryption library
HE key management
Recieves a public key or a signature
Receives private keys
IoT device
Publish the public encryption key
Publish public signing keys
Blockchain
Receive the public encryption key and public signing keys
ARCADIAN IoT Framework

Figure 51 - Logical architecture view of HE

- HE library is a software library developed in Go language with binding to Python, Java, C, JS that can be deployed on ARCADIAN-IoT devices, so that they can secure their data at rest or data to be transmitted. The library provides a simple API to encrypt and decrypt data with the ABE paradigm, sign the encrypted data with SIM based RoT signatures, verify the signatures, and interface with other components to obtain private or public cryptographic material.
- Middleware is a connecting subcomponent allowing the HE library to request predefined actions from the SIM subcomponent through the modem (via AT commands)
- ARCADIAN-IoT SIM profile will live in a hardware secure element, having a security applet allowing to preform actions with guarantied security (have as baseline GSMA IoT SAFE specifications). In particular, the implemented applet enables generating private and public cryptographic keys, where the private keys remain secured in the SIM (the SIM provides no way to retrieve these secrets). Furthermore, the applet can use the generated keys to cryptographically sign data with RoT information.
- HE key management is a component handling key distribution. It can be deployed as a decentralized system. To enhance trustworthiness, it publishes the public material on the blockchain to guarantee that it was not maliciously modified. Furthermore, it defines distribution rules for the private keys based on the ARCADIAN-IoT Authentication component.

4.1.3.2 Sub-use cases

The main use case of the HE component is to enable efficient encryption/decryption with RoT information that can be applied to as many IoT scenarios as possible: for example, it will be evaluated in an emergency and vigilance service with drones, industrial IoT devices and with medical IoT data. Besides the latter main use case, that suffices for a standard, non-compromised working of the ARCADIAN-IoT platform, we also report on the following sub-use case invoked in the case of a security incident.

4.1.3.2.1 SIM security applet and HE in device self-protection and device-self recovery

ARCADIAN-IoT security applet (and its action within HE component) will be used as well for the device self-protection and self-recovery. Considering its independence of the device itself (it has its own hardware and software inside the secure element) and its means to be securely reached only by the network operator, it is in a unique positioning to act in situations of compromised non-cooperative devices. In the final prototype developed for the project, the security applet is already

ready for receiving, through proven secure GSM over-the-air services, information about the trustworthiness of the device where the SIM is. When receiving this information, the security applet prototype has implemented 2 functionalities that go beyond any known state of the art (as the HE component already does):

1. If the device trustworthiness level indicates that the device is compromised, the SIM security applet will refuse to use its cryptographic processes to sign HE payloads. Complementing other protection measures, e.g., based in cellular networks authorization processes, this will ensure that a compromised device, that could use other forms of communication to reach ARCADIAN-IoT components (e.g. WiFI), will not be able to deceive any ARCADIAN-IoT component that receives its payloads, because these will not be signed by the RoT. By not signing the payloads, the security applet is also informing ARCADIAN-IoT components deployed in the device that the device is found to be compromised, and these components can act accordingly.
2. Complementing the previous action, when a device is found to be trustworthy after a period of being compromised, the SIM security applet also has means to be informed and act upon this information. In this case its regular operational behaviour within the HE will resume. This action allows other ARCADIAN-IoT components living at the device to be informed and to recover their normal behaviour as well.

These security measures are triggered by ARCADIAN-IoT network-based authorization enforcement component (described in D4.3). This component, that lives in the cellular network operator core infrastructure, will also act upon devices trustworthiness information, and therefore it is in a unique position to distribute trustworthiness information to the devices' hardware secure element (the uniqueness of the position is due to the access to the security information and the means to reach SIMs according to GSM proven secure standards).

4.1.3.3 Sequence diagrams

The sequence diagram in Figure 52 explains how the HE subcomponents for the ABE encryption and SIM signatures interact during the onboarding of a device, encryption, and decryption process. As one can observe in the sequence diagram, during the onboarding phase the HE library obtains from the SIM a public signing key, that is then confirmed by the key management and published on the permissioned blockchain. On the other hand, the private keys are obtained from the HE key management where the device needs to authenticate itself. In the phase of encryption, the library encrypts data and uses SIM to obtain RoT signatures. In the decryption phase the encrypted data is first verified, where the public key is checked ether by checking the certificate or by checking the permissioned blockchain, and then decrypted.

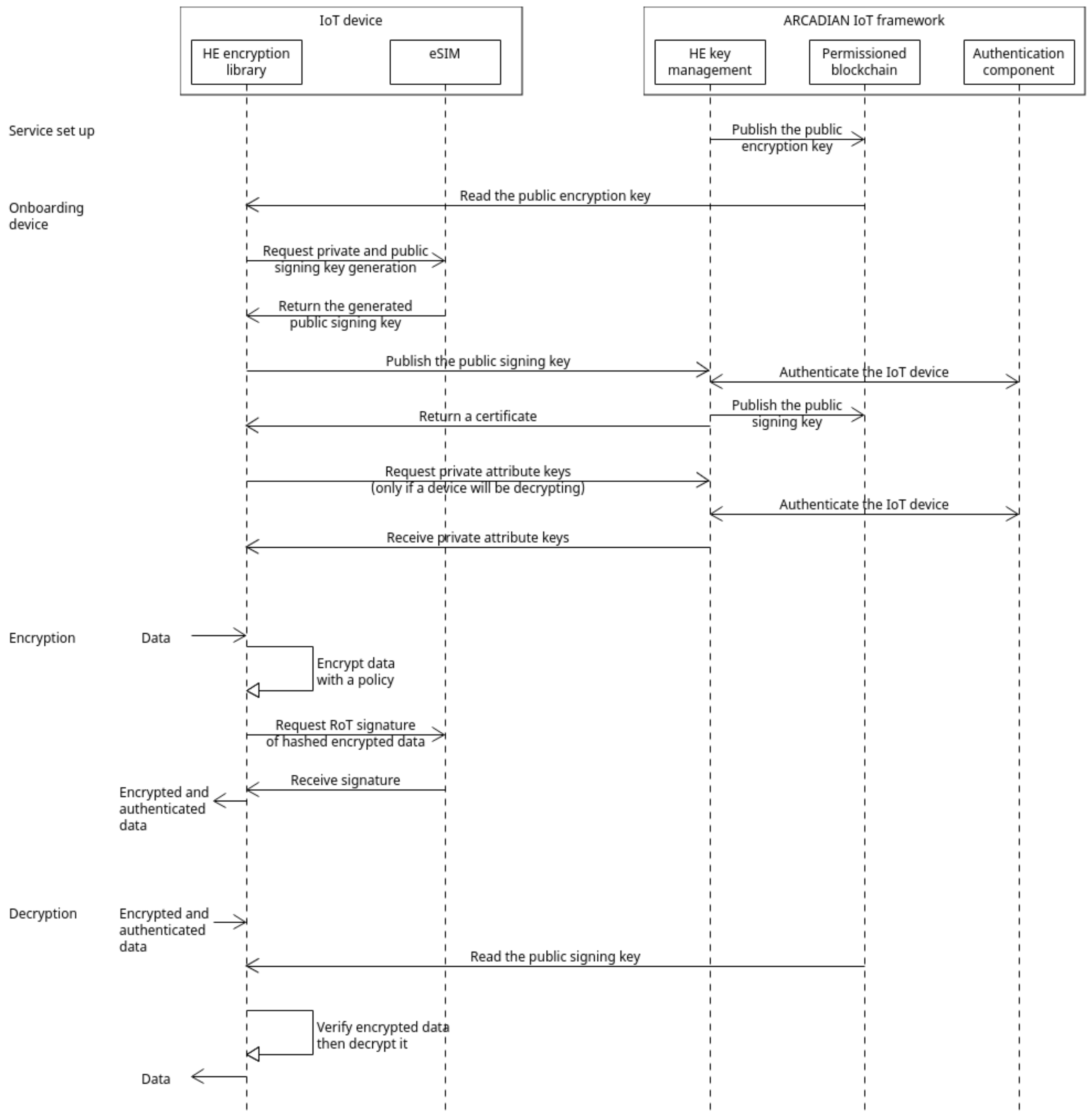

IoT device
HE encryption library
eSIM
ARCADIAN IoT framework
HE key management
Permissioned blockchain
Authentication component
Service set up
Publish the public encryption key
Onboarding device
Read the public encryption key
Request private and public signing key generation
Return the generated public signing key
Publish the public signing key
Authenticate the IoT device
Return a certificate
Publish the public signing key
Request private attribute keys (only if a device will be decrypting)
Authenticate the IoT device
Receive private attribute keys
Encryption
Data
Encrypt data with a policy
Request RoT signature of hashed encrypted data
Receive signature
Encrypted and authenticated data
Decryption
Encrypted and authenticated data
Read the public signing key
Verify encrypted data then decrypt it
Data

Figure 52 - The sequence diagram of the HE component with internal and external interfaces

4.1.3.4 Interface description

The HE subcomponents interface with external ARCADIAN-IoT components in the following way:

- HE key management uses the Permissioned blockchain component to publish public cryptographic material. This adds additional trust in the system anchoring it on a decentralized ledger. In particular, the public ABE key of each key generation authority is published, as well as the public signature verification keys, see Section 4.2 for the details.
- HE key management depends on the Authentication component to ensure that the private cryptographic keys are distributed to the appropriate entities. In particular, each device or person will need to authenticate itself to the HE key management authorities to obtain the needed keys for the access/decryption of data.
- HE key management sends information about private key requests to the Reputation

system that can use it to modify the trust score of a client in cases of malicious behaviour.

- HE component is used, besides by the devices in the framework, by other ARCADIAN-IoT components:
    - Remote Attestation component requires the attesters to use the HE encryption library (and through it SIM) to securely respond to an attestation event with private claims and hardware-based RoT signatures. Furthermore, the Remote Attestation component uses the HE key management to obtain appropriate keys for the attestation procedure.
    - Self-aware Data Privacy component uses HE encryption library to enforce privacy policies by encrypting the data in a database.
    - Self-recovery component uses HE to encrypt the data that is stored for recovery with a policy that allows appropriate recovery authorities to decrypt it and perform the recovery.
    - Biometrics component uses HE component to securely store and transmit private biometrical data.

4.1.3.5 Technical solution

4.1.3.5.1 Deployment architecture view

The HE components are meant to be deployed in the following way:

- HE encryption library with RoT signatures needs to be deployed (installed) on any device that performs encryption/decryption/signature operations. In the case of ARCADIAN-IoT, these are IoT devices such as drones, phones, servers, web applications running in browsers, and other ARCADIAN-IoT services. Note that if the device does not have SIM capabilities, it can still use HE library for encryption and decryption where signatures do not have a hardware-based RoT properties.
- HE key management is deployed as part of the ARCADIAN-IoT service. In particular, it is a Dockerized service that can be deployed on one or multiple (decentralized) servers. It can be accessed using REST calls, see the following section.

4.1.3.5.2 API specification

In this section we detail the interface to the functionalities of HE.

HE library

We provide APIs for multiple programming languages, while in this section we focus only on the use of the library with Python bindings, since other interfaces are similar.

Installation

The core of the library is implemented in Go. We have cross-compiled the shared objects (e.g., `libahe.so`, `libahe.h`) that need to be downloaded before the use of the Python bindings. To install the Python library, we provide an automated installation flow with `pip`.

The encryption part of the library is a pure software implementation and has been tested on AMD as well as ARM processors. On the other hand, RoT signatures are provided by the SIM security applet through the device middleware provided for that purpose. The security applet needs to be in a SIM profile, which it is placed in the devices when the SIM with it is provisioned or the SIM card with it is inserted in the device. It can also be provisioned Over the Air at the device at any moment, according to eSIM/iSIM-related GSMA specifications. In this project, the device middleware is compiled jointly with the remaining code from the HE component.

HE library use

Assuming the library has been installed, a device can start using it functionalities. First the library needs to be imported:

```
from ahe-bindings import Ahe

# initiate
g = Ahe("/path/to/libahe.so")
maabe = g.NewMaabe()
```

Before a device can sign payloads, it needs to create its signing key in its SIM, and output the verification key. If the device has no embedded element, then it cannot produce RoT signatures and this step is skipped.

```
# initiate an entity by generating its signing key (in the eSIM)
# and verification key
verification_key = g.GenerateSigningKeys()
```

One can specify the access policy and use the library to encrypt data:

```
# encrypt a message with a decryption policy
msg = "Attack at dawn!"
bf = "((auth1:at1 AND auth2:at1) OR (auth1:at2 AND auth2:at2)) OR (auth3:at1 AND auth3:at2)"
enc = g.Encrypt(maabe, msg, bf, pks)
```

The above assumes that a device has obtained public keys (the value `*pks*` above) from the key management authorities. The policy can be specified with a Boolean formula that determines which attributes are needed to decrypt a message. For example, one could replace in the above 'auth1:at1` (i.e., the attribute 1 that has been created by authority 1) with the unique name of a device used in ARCADIAN-IoT framework, or a group of receivers of the message, e.g.; system administrators, etc.

Assuming that the device has a SIM with ARCADIAN-IoT security applet in it, it can sign the encrypted data. If multiple ciphertexts were created with different access policies, they can be signed together.

```
# sign a message with RoT signature
enc_signed = g.Sign([enc])
```

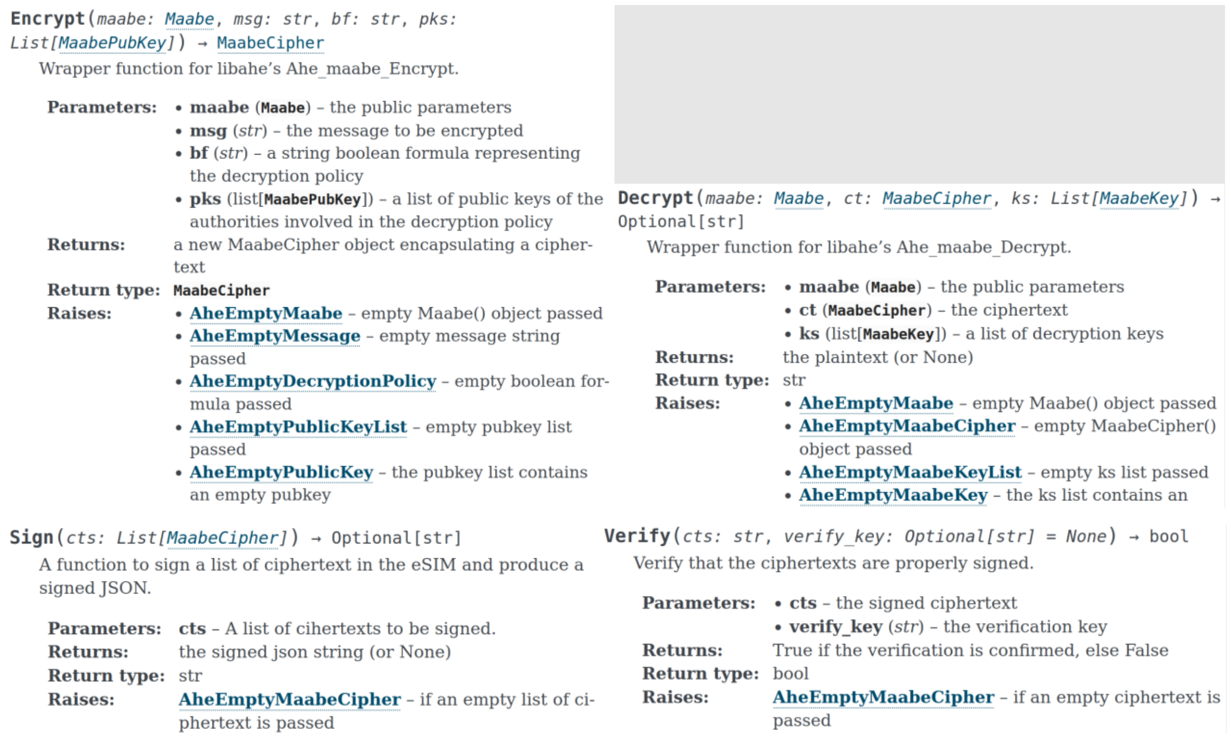
A receiving device can validate the authenticity of data (directly on the ciphertext)

```
# verify the authenticity of the ciphertext
check = g.Verify(enc_signed, verification_key)
```

and if it has appropriate attribute keys (the value `*ks1*` below), it can decrypt the corresponding ciphertexts:

```
pt1 = g.Decrypt(maabe, enc, ks1) # returns msg
```

If the device does not possess the access rights, an error is raised. In Figure 53 more documentation is available.



Encrypt(*maabe: Maabe, msg: str, bf: str, pks: List[MaabePubKey]*) → MaabeCipher

Wrapper function for libahe's Ahe_maabe_Encrypt.

Parameters:
- **maabe** (`Maabe`) – the public parameters
- **msg** (*str*) – the message to be encrypted
- **bf** (*str*) – a string boolean formula representing the decryption policy
- **pks** (list[`MaabePubKey`]) – a list of public keys of the authorities involved in the decryption policy

Returns: a new MaabeCipher object encapsulating a ciphertext

Return type: `MaabeCipher`

Raises:
- **AheEmptyMaabe** – empty Maabe() object passed
- **AheEmptyMessage** – empty message string passed
- **AheEmptyDecryptionPolicy** – empty boolean formula passed
- **AheEmptyPublicKeyList** – empty pubkey list passed
- **AheEmptyPublicKey** – the pubkey list contains an empty pubkey

Decrypt(*maabe: Maabe, ct: MaabeCipher, ks: List[MaabeKey]*) → Optional[str]

Wrapper function for libahe's Ahe_maabe_Decrypt.

Parameters:
- **maabe** (`Maabe`) – the public parameters
- **ct** (`MaabeCipher`) – the ciphertext
- **ks** (list[`MaabeKey`]) – a list of decryption keys

Returns: the plaintext (or None)

Return type: str

Raises:
- **AheEmptyMaabe** – empty Maabe() object passed
- **AheEmptyMaabeCipher** – empty MaabeCipher() object passed
- **AheEmptyMaabeKeyList** – empty ks list passed
- **AheEmptyMaabeKey** – the ks list contains an

Sign(*cts: List[MaabeCipher]*) → Optional[str]

A function to sign a list of ciphertext in the eSIM and produce a signed JSON.

Parameters: **cts** – A list of cihertexts to be signed.

Returns: the signed json string (or None)

Return type: str

Raises: **AheEmptyMaabeCipher** – if an empty list of ciphertext is passed

Verify(*cts: str, verify_key: Optional[str] = None*) → bool

Verify that the ciphertexts are properly signed.

Parameters:
- **cts** – the signed ciphertext
- **verify_key** (*str*) – the verification key

Returns: True if the verification is confirmed, else False

Return type: bool

Raises: **AheEmptyMaabeCipher** – if an empty ciphertext is passed

Figure 53 - Snippets from the HE documentation

HE key management

This subcomponent is deployed as a web service that can be accessed by REST calls.

- Requesting public ABE key: The public ABE key can be obtained by a GET request to arcadianHEaddress/pubkeys. The response is a JSON representing the public key of the ABE scheme. Note that the public key can be requested also from the Permissioned blockchain or in the case of decentralized key management, from multiple authorities (checking a consistent response). HE library includes functionalities to unmarshall the public key to an object that can be used by encryption functionality.
- Requesting private ABE keys: Private attribute keys can be obtained by a POST request to arcadianHEaddress/get-attribute-keys. It expects to submit a JSON with fields *uuid* (universal unique identifier of the client), *attribs* (an array of attributes for which the client request private keys) and *token* (an authentication token proving the identity of the client). The server responds with a JSON representing a private key. In the case the key management service is decentralized, the request needs to be sent to all of the key management servers, which initiate an MPC protocol and deliver a private key. HE library includes functionalities to unmarshall the private key to an object that can be used by decryption functionality.
- Publishing public signature key: A client can publish its public signature key by a POST request to arcadianHEaddress/pub-signature-keys. It expects to submit a JSON with fields *uuid* (universal unique identifier of the client), *verkey* (public signature key, i.e. verification key) and *token* (an authentication token proving the identity of the client). The server responds with a JSON representing a certificate proving that the public verification key truly belongs to the client.

4.1.3.5.3 Security aspects

The ABE technology is relatively new and has been deployed in relatively small number of projects. Nevertheless, the security of it has been cryptographically proven to depend on well-established cryptographical assumptions, similarly as the everyday used standard public key schemes. In particular, the implemented ABE schemes depend on the hardness of finding a discrete logarithm is certain elliptic (pairing) groups, known as BN256. This is a standard assumption, whose main threat in the future is the development of a quantum computer (as it is with the majority of commonly used cryptography). Enhancing the system to a quantumly secure one is possible but would majorly decrease the efficiency.

The SIM technology is used for several decades for protecting subscribers' identity and the key material that allows to authenticate and attach to cellular networks. This technology continues to be well-accepted as secure. While there may be security optimizations to ARCADIAN-IoT's SIM security applet, its main functionality is assumed to be secure as well. The secrets used to sign the encrypted payloads are generated in the secure element and never leave it (just the public keys are provided to allow the signature validation). This happens because there is no known form of extracting the secrets from the SIM secure element, particularly if it is an embedded or integrated form (which makes it almost impossible to clone the secure element).

4.1.4 Evaluation and results

In this section we report on the main outcomes and results of the evaluation of the Hardened Encryption supported by SIM as RoT.

KPI1: Provide an encryption library enabling fine-grained access control over data with APIs in at least 4 programming languages and demonstrate its use on at least 2 types of devices/platforms.

As stated in the previous sections, one of the produced resources is an encryption/decryption and signature library enabling the use of ABE for fine-grained control over data. The library has APIs in 5 programming languages: besides Go that was chosen for the implementation of the core functions, it has bindings to Python, Java, JS, and C. The library was compiled for multiple devices and its use was demonstrated on Android phones, Linux based servers, the library was integrated to run directly in a web-browser of a client using a web-service, and multiple models of Raspberry Pi. The tests of the library were done as part of WP3 development process, as well as WP5 component validation.

KPI2: Add Root of Trust information to encrypted data with SIM based signatures.
As part of work in T3.2, a mechanism for hardware-based RoT was developed. As described above, this includes a release of an applet running on SIM cards (and SIM form, like SIM, eSIM or iSIM), as well as the development of the middleware that enables a user's device to communicate with the it. In fact, two versions of the middleware were developed:

- Python middleware supporting RoT signatures in Linux-based IoT devices with an attached SIM, such as the Drone being used in the project, Raspberry Pi based devices, etc.
- Java based middleware for RoT signatures in Android devices

Both middlewares were integrated in the beforementioned encryption library with unified interface. In fact, the library enables to use non-hardware-based signatures and upgrade to RoT signatures with minimal effort/change (assuming the device has an SIM).

KPI3: Provide a secure and scalable key management and delegation synchronized with the decentralized identity management, multi-factor authentication and self-recovery.

As described in Section 4.1.2.2.2 we have implemented key management systems for both FAME and MAABE schemes. Moreover, we have put an additional effort to decentralize the FAME key management using MPC. We have tested it to handle hundreds of simultaneous key requests. Moreover, we have provided Docker scripts to simplify the deployment. The key management was integrated with ARCADIAN-IoT identity management and multi-factor authentication to define and enforce access rights to the users of the system. Moreover, the ARCADIAN-IoT Self-recovery

component uses the system to allow secure distribution of recovery access.
In Table 20 we report on the response times needed for the (network of) key servers to return a requested private key, corresponding to attributes of ABE. All the tests were performed over LAN. All the response times are acceptable for a web-communication process.

Key management response time	1 attribute	5 attributes	10 attributes	15 attributes	20 attributes
FAME centralized key management	51ms	70ms	98ms	112ms	183ms
FAME decentralized key management with 3 MPC nodes	212ms	221ms	359ms	583ms	679ms
MAABE management (response time of one authority)	5ms	7ms	11ms	15ms	17ms

Table 20 Response time of the HE key managements

KPI4: Provide efficient implementation of the process with encryption times comparable to the state-of-the-art results on multiple devices, and with RoT signatures in acceptable time for the communication processes.

The main reason for why we chose to develop the encryption library from a Go-based core, was that the existent cryptographic libraries in Go provide a secure and performance-wise optimized basis for the implementation of high-level protocols. This helped us achieve fast evaluation of functionalities offered by the library. Furthermore, Go can be compiled to binaries that can be used by other programming languages, without a significant efficiency slowdown. In fact, we have tested this claim with our implementation and observed no major slowdown when using, say, Python in comparison to Go. Below we report the average encryption and decryption times using the HE library. Tests have been performed on a laptop with 1.70GHz AMD Ryzen 7 PRO 4750U, an Android phone Samsung A33 with 2.4 GHz Cortex-A78 processor, and on a Raspberry Pi 3 Model B+ with 1.4GHz Cortex-A53. Note that the processes depend on the complexity of the access policy, where more attributes used result in higher times. As explained in Section 4.1.2.2.1, we are using the ABE scheme to encrypt a random key, that is then further used for encrypting data with a symmetric encryption scheme. Hence, we report in Tables 15, 16 only on the overhead created by the ABE, while the performance of symmetric schemes, such as block ciphers using AES, is well understood, optimized and depends on the size of encrypted/decrypted data.

MAABE scheme	1 attribute	5 attributes	10 attributes	15 attributes	20 attributes
Encryption (laptop)	4ms	14ms	30ms	43ms	58ms
Decryption (laptop)	4ms	10ms	22ms	30ms	41ms
Encryption (phone)	9ms	26ms	46ms	68ms	90ms

Decryption (phone)	6ms	23ms	62ms	115ms	185ms
Encryption (Raspberry Pi)	1.2s	4.7s	8.7s	12.9s	16.9s
Decryption (Raspberry Pi)	0.6s	3.1s	6.2s	9.3s	12.5s

Table 21 - Encryption and decryption overhead for using MAABE scheme

FAME scheme	1 attribute	5 attributes	10 attributes	15 attributes	20 attributes
Encryption (laptop)	6ms	34ms	117ms	236ms	436ms
Decryption (laptop)	7ms	7ms	7ms	8ms	8ms
Encryption (phone)	19ms	52ms	169ms	352ms	597ms
Decryption (phone)	19ms	20ms	20ms	21ms	21ms
Encryption (Raspberry Pi)	1.2s	4.7s	8.7s	12.9s	16.9s
Decryption (Raspberry Pi)	0.6s	3.1s	6.2s	9.3s	12.5s

Table 22 - Encryption and decryption overhead for using FAME scheme.

To argue that we have achieved the KPI, we compare our results to the state-of-the art. For the main reference we take [27], where an ABE scheme was implemented in Java and benchmarked on a laptop and an android phone. the following computation times were reported:

- Encryption:
    - A laptop with 1.60GHz Intel Quad-Core i7: ~160ms for a policy with 5 attributes
    - An Android phone with 1.60GHz Intel Atom Z2460: ~2.5s for a policy with 5 attributes
- Decryption:
    - A laptop with 1.60GHz Intel Quad-Core i7: ~250ms for a policy with 5 attributes
    - An Android phone with 1.60GHz Intel Atom Z2460: ~6s for a policy with 5 attributes

Firstly, we note that a direct one-to-one comparison between our results and [28] cannot be made, since we are using different ABE schemes and benchmarking it on a slightly more performant machine. Nevertheless, the performance numbers were improved and indicating that the technology can be practical in IoT scenarios. The improvements could be attributed to the following. On one hand we used a programming language that can be compiled for the specific devices which gives a better performance than Java used in the reference benchmarks. Secondly, the cryptographic libraries, that we used in the implementation and are using elliptic curves as the underlying mathematical object, have been improved in the recent years. Moreover, the cryptographic schemes have been improved to a better efficiency as well. In particular, the MAABE scheme accelerates on the encryption while the complexity of the FAME's decryption does not increase majorly when decrypting ciphertexts with complex policies. In the case of applying the scheme on phones, we also point out major hardware improvements making them almost as powerful as laptops. On the downside, it seems that the applicability of ABE technology on devices with limited capabilities is still bounded to use-cases where relatively simple policies are applied.

As for the SIM RoT signature process and the KPI "add Root of Trust information to encrypted data with SIM based signatures", it is known that the computing power of any SIM form is limited. This was a concern since the beginning of the project and, in fact, the research hypothesis formulation considered exactly this when 1GLOBAL and XLAB decided that the SIM should sign the hash (SHA256) of the encrypted payload and not the encrypted payload itself. This allowed to perform a process with very low computational effort in the SIM, allowing it to act as Root of Trust by digitally signing the hash of the payloads sent by the devices.
The final prototype was assessed with IoT solution providers of the consortium, within their own solutions, and no concerns related with the SIM performance were raised. More specifically, and in lab, tests to the ARCADIAN-IoT security SIM applet were done with (1) a SIM card reader (Mitsai MCR3334BK), (2) a Raspberry-Pi 3 and (3) an Android device – Samsung S22. With these devices, were also tested 2 different SIMs from different manufacturers: STMicroelectronics ST4SIM-200M (STM) and EastCommPeace ST33G1M2 (ECP).
The performance assessment was done for each of the several procedures needed, for each device and SIM manufacturer. As the objective was to have "RoT signatures in acceptable time for the communication processes" are presented next the worst and best cases from the myriad of devices and SIMs tested (values represent the average of 10 executions of the process in the device and SIM mentioned).

- **Security Applet Selection**: The worst case is of 2936,6ms, in the card reader with the ECP SIM. In the same device with the STM SIM the applet selection would take just 807,9ms.
- **Generate Key Pair**: The worst case is of 454,3ms in the Android device with the ECP SIM. The best time was found in the Raspberry Pi with the STM SIM: 192,7ms.
- **Compute Signature Init**: The worst case is of 35,1ms in the Android device with the STM card. The best time gathered was in the card reader with the ECP SIM: 12,5ms.
- **Compute Signature Update**: The worst case is of 447,9ms in the Raspberry pi with the ECP card. The best time was of 110,7ms, in the same device with the STM SIM.

It is important to understand that the signature is performed with the *Compute Signature Update*, and that the remaining commands are of applet selection and process initialization. Therefore, what is expected to be more relevant in terms of performance is exactly the process of performing the signature, as the other commands are not executed often. In the worst-case scenario of the tests made, performing a signature of a payload would take less than half a second – in the best-case scenario it would take ~0,1 seconds. This led to the assessment made by the IoT solution providers, that the SIM performance was not compromising the performance of the functionalities of their solution.

4.1.5 Future work

In what regards the use of SIM as Root of Trust, 1GLOBAL envisions to start exploring the technology commercially with its IoT clients, coupling its connectivity products with these new security features.

4.2 Hardened Encryption with cryptochip

4.2.1 Overview

4.2.1.1 Description

The Hardened Encryption component with a crypto chip system has been designed and tailored specifically for industrial scenarios where IoT devices have limited or restricted capabilities. This system, on one hand, includes crypto chips (as add-on module or embedded into an IoT device) enabling that the IoT device's master firmware is equipped with a firmware agent for handling the keys. On the other hand, the system also includes the IoT middleware software platform (specific for the selected scenarios); this platform incorporates features for (i) key management of crypto chip keys, (ii) ARCADIAN-IoT integrated security systems management as end point, (iii) a software agent (running into Middleware software – cloud side) for handling the decentralized authorization (as a second protection during authorisation, on top of hardware encryption) and (iv) interface relaying with 3PP platforms by API and RabbitMQ.

4.2.1.2 Requirements

A recall of the high-level requirements that have been previously defined and provided in deliverable D2.4 [2] is given below.

Requirement 4.1.1 – Encryption mechanism: Enable secure and lightweight encryption with access policy strengthened with hardware-based RoT information.

Requirement 4.1.2 – Secure key-generation: Provide secure and scalable key management and delegation synchronized with the decentralized identity management.

4.2.1.3 Objectives and KPIs

With the aim of fulfilling the project's objectives, the Hardened Encryption component will be evaluated against the following KPIs that are related to the main project objective "Provide a Hardened Encryption with recovery ability" (as recalled in the introduction).

KPI scope	
Provide secure and scalable key management, in both device & middleware sides, covering all operations lifecycle situations (creation, fulfilment, regular data communications, removal, recovery) for devices fleets.	
Measurable Indicator	
Simplicity of process, respecting the security conditions. Agnostic management operations capability (applicable to manipulation of crypto chip producer firmware into device side, applicable to any cloud platform enclosing the middleware component into the cloud side).	
Benchmarking	
Similar security products provisioning methods and tools.	
Target value	***Achieved value***
Less than 5min provisioning (for any scenario) into device side, with GUI in place and how to procedure, into device side.	5 min provisioning (for any scenario) into device side, performed by an experienced user familiar with device firmware GUI.

Less than 2min provisioning (for any scenario) with web dedicated page in place and how to procedure, into middleware side.	2 min provisioning (for any scenario) into middleware side, performed by an experienced user familiar with middleware GUI.

KPI scope	
Provide efficient implementation of the process at a device and the middleware side in acceptable time for the communication (telecommunication protocols and upper layer OSI protocols wise) processes.	
Measurable Indicator	
T1_E – Time duration between sensors data stream aggregation and encrypted payload generation, by device firmware agent designed and build for encryption. This indicator is applicable for sense device – to – IoT platform. T4_D – Time duration between encrypted payload receiving and actuators “in clear” commands, by same device firmware agent. This indicator is applicable for sense IoT platform – to – device. T3_E – Time duration between encrypted TLS payload received from IoT platform, decryption by certificate applied, and encryption with correspondent hardware key of the payload, by dedicated local middleware agent. This indicator is applicable for sense IoT platform – to – device. T2_D – Time duration between receiving the encrypted payload received from device and decryption by correspondent hardware key of the payload, by dedicated local middleware agent, and relaying forward by TLS to IoT platform. This indicator is applicable for sense device – to – IoT platform.	
Target value	***Achieved value***
T1_E lower than 2sec T4_D lower than 6sec T3_E lower than 4sec T2_D lower than 8sec	T1_E lower than 2sec T4_D lower than 5sec T3_E lower than 4sec T2_D lower than 7sec

4.2.2 Technology research

4.2.2.1 Background

4.2.2.1.1 Crypto chip System

The usage of crypto chips for hardened encryption processes is particularly pertinent for industrial IoT devices, where computing power capacity is limited and unable to run TLS type certificates, while security of both the transmitted and received streams of data from / to field sensors and actuators and the corresponding data management platforms is crucial. The proposed crypto chip-enabled hardened encryption solution provides robust, scalable, application-agnostic security with both crypto chip and device vendor neutrality. Besides the core feature of symmetric encryption, it is being designed to integrate with / leverage from the additional security features from other ARCADIAN-IoT components, namely Authentication, Remote Attestation, Reputation System, Network Base Authentication, Self-Aware Data Privacy and Behaviour Monitoring. The solution is aimed at being highly scalable, easy to be implemented with any type of IoT platform and by any IoT device vendor and resolves main cyber security aspects for grid & other critical commercial verticals, relating to industry 4.0. For the currently prototyped industrial IoT device, a crypto chip designed by STMicroelectronics was chosen, which is by default an authority for issuing Root of Trust products. To confirm completely the chain of trust, middleware software component of Hardened Encryption / crypto chip system is planned to be tested by a penetration testing formal certified entity.

4.2.2.2 Research findings and achievements

4.2.2.2.1 Hardened encryption with crypto chip system

In this section we report on the design of the crypto chip-enabled Hardened Encryption, including subcomponent's design, its critical challenges (that were passed successfully) and the correspondent achievements. The component consists of following parts: hardware designed for the secure communication of otherwise computationally limited devices, and the middleware communicating with the devices.

4.2.2.2.2 Hardware challenges and achievements

The following components have been developed:

- 1 industrial IoT device motherboard, with crypto chip working as an electronic sub system (dedicated power supply and power management, communication paths and channels with motherboard microcontroller); board was finally successfully designed, tested in simulator, executed in a small series, and tested practically with electronic tools under a large set of scenarios (continuity, working stages, abnormal interruptions, unstable power supply, etc.); main challenge stayed with designing a board which can easily adapt any crypto chip model, without large redesigning / redeployment; practically, BOX2M innovates a way for fast adaption of crypto chips into MCU powered IoT devices
- 3 extension boards, running the grid interfaces roles, for data gathering (2 different boards) and OTA / remote / remote commands (1 dedicated board); all boards were similarly tested individually, but also integrated electronically with motherboard, to test the whole integrated system (as power management and communication internal buses), using specific real sensors and actuators, into BOX2M laboratory; main challenge stayed with designing from scratch a robust plug-in system, solid enough to support cooperation with a mother board with embedded security, and be focused on grid & utilities domain, mainly, by sensors type interfaces
- 9 communication boards (GSM, UMTS, LTE, LTE-M, NBIoT, 5G, ETH, WiFi, ETH & WiFi combined), from which 8 were completely designed and executed (5G board deployment was aborted, due to an errored technical data sheet provided by 5G radio module vendor, which generated a not working board – by wrong antennas layout and number), All 8 were tested completely; all mobile broadband boards have by default an xSIM too, so tests were done with both a regular SIM and an xSIM; on top of regular QoS for telecommunication, it is important to validate, with each communication system, impact of KPIes (because total end-user time, which operates data from IoT platform, on both ways (from device to IoT platform, and viceversa) are including not just the pure encryption and decryption times (KPIes T1_E, T4_D, T2_D, T3_E), but also the telecommunication duration for carrying the encrypted data between device and cloud hosting the Docker containing the middleware as encryption system component; challenges were to release mobile broadband boards with both xSIM and regular SIM, and to manage balanced the power supply management and internal communication channels between motherboards and telecommunication systems
- Specific hardware testing simulators were designed and built into lab, and used intensively during project lifecycle, e.g. multiple sensors blocks (from various vendors, various models), actuation field elements blocks, mobile kits for drive testing

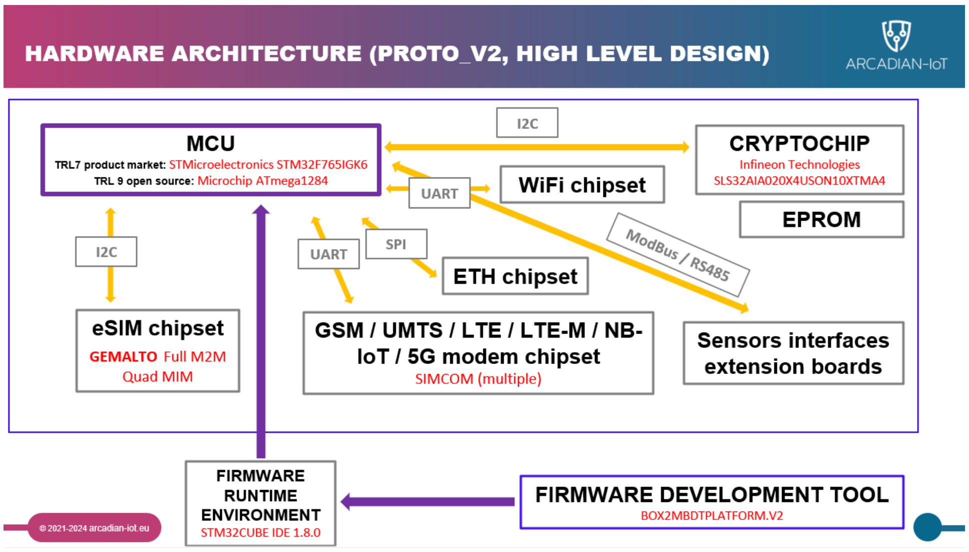

HARDWARE ARCHITECTURE (PROTO_V2, HIGH LEVEL DESIGN)
ARCADIAN-IoT
MCU
TRL7 product market: STMicroelectronics STM32F765IGK6
TRL 9 open source: Microchip ATmega1284
I2C
CRYPTOCHIP
Infineon Technologies
SLS32AIA020X4USON10XTMA4
UART
WiFi chipset
EPROM
ModBus / RS485
UART
SPI
ETH chipset
I2C
eSIM chipset
GEMALTO Full M2M Quad MIM
GSM / UMTS / LTE / LTE-M / NB-IoT / 5G modem chipset
SIMCOM (multiple)
Sensors interfaces extension boards
FIRMWARE RUNTIME ENVIRONMENT
STM32CUBE IDE 1.8.0
FIRMWARE DEVELOPMENT TOOL
BOX2MBDTPLATFORM.V2
© 2021-2024 arcadian-iot.eu

Figure 54 - Hardware high level architecture

4.2.2.2.3 *Firmware challenges and achievements*

The main challenge was to deploy a master firmware system, accommodating into a MCU loaded master structure, subsystems firmware, dedicated for each technology aspect: boards wholistic management, data management, encryption and decryption agent by adapting the supplying crypto chip vendor library and keys repository, firmware of each telecommunication module. Challenge was successfully passed, firmware was designed, deployed and tested, ending with an end-to-end encryption and decryption operations pointing an IoT platform and running all life cycle scenarios. For edge computing agents, network management agents, telemetry agents and sensors data gathering agents, we have reused part of existing registered under intellectual property of BOX2M "M0 firmware". Encryption & decryption system device component (from firmware perspective) can be integrated in any other device vendor firmware, and can adapt crypto chips of the same type, or from different vendors, by adapting their working libraries.

Due to complexity, it was needed to design and deploy a CLI (command line interface) tool, to operate faster firmware changes, especially during long testing processes.

4.2.2.2.4 *Middleware challenges and achievements*

The main challenges were to deploy the subcomponent as a modular component, Docker contained, configurable API and RabbitMQ ready with any 3PP system, including the integration with ARCADIAN-IoT security systems used by Hardened Encryption such as Remote Attestation, Reputation System, Self-aware Data Privacy, Decentralized authentication. Another challenge was the fast integration with a live IoT platform, in order to have the capability to run whole experiments proposed. These solved challenges helped us practically to achieve first 2 main deliveries, respectively encryption & decryption system (both ways) and API readiness to continue the specific ARCADIAN-IoT systems integration. These great achievements could not had been done without firstly securing a proper hardware component and firmware design and deployment, which shows, from applied research perspective, that without knowing in detail the domain of application, it cannot be released an effective product to address / resolve field problems, as it is the grid security on microcontroller powered devices. Middleware was stressed into a live IoT traffic, generated by BOX2M lab setup, with real sensors and using as prefab payload thousands of parameters edge collected from these, to assess the volume scalability and measure times for encryption and decryption operations. For telemetry, Docker orchestration and TLS interfacing, we have reused part of existing registered under intellectual property of BOX2M "Industrial Telemetry Platform". Middleware included a front end for lifecycle operations of provisioned assets and a service operation center, with events and alerts configurable for managed assets.

To better test the integration with other security system, we had to design and deploy an IoT devices software simulator, in scope of simulation of fleets of devices.

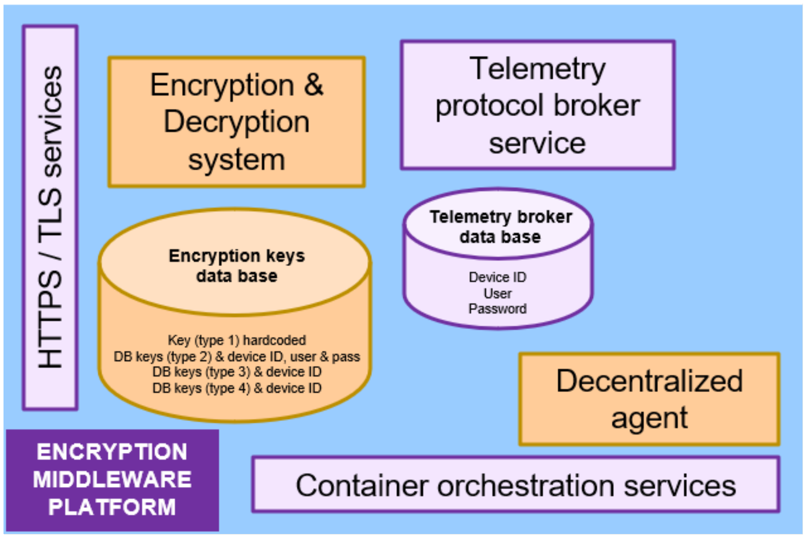

HTTPS / TLS services
Encryption & Decryption system
Telemetry protocol broker service
Encryption keys data base
Key (type 1) hardcoded
DB keys (type 2) & device ID, user & pass
DB keys (type 3) & device ID
DB keys (type 4) & device ID
Telemetry broker data base
Device ID
User
Password
Decentralized agent
ENCRYPTION MIDDLEWARE PLATFORM
Container orchestration services

Figure 55 - Middleware architecture

4.2.2.3 Produced resources

In this section we report the implementation of the HE component and the functionalities of the final prototype. With respect to the crypto chip-enabled hardened encryption, the highlighted produced resources are:

Hardware:

- Industrial IoT hardware device (see Figure 54 and Figure 56)
- A complete IoT stress test environment (hardware set-up lab, cloud set-up, service operation centre, defined measurements, IoT devices simulator)

Firmware and software – private (in scope of IP registration):

IP registration as “M6 firmware” ongoing:

- Device firmware & its CLI application for management
- Encryption & decryption firmware agent

IP registration as “Hybrid Encryption Middleware” ongoing:

- Middleware software application
- Encryption & decryption middleware agent
- API configurator

Software – public (shared with consortium partners):

- API integration with 2 live IoT platforms (BOX2M and Martel), to have the test bed operational end-to-end, and to demonstrate versatility of security system integration using 2 different methods (API and RabbitMQ), e.g. Self-Aware Data Privacy
- Integration with all involved security systems, tested and validated, via RabbitMQ interface

Reference link: BOX2M_API & RabbitMQ - integration documentation.pdf

under WP5 – Task 5.2 - 5.4 Use cases implementation – Domain B

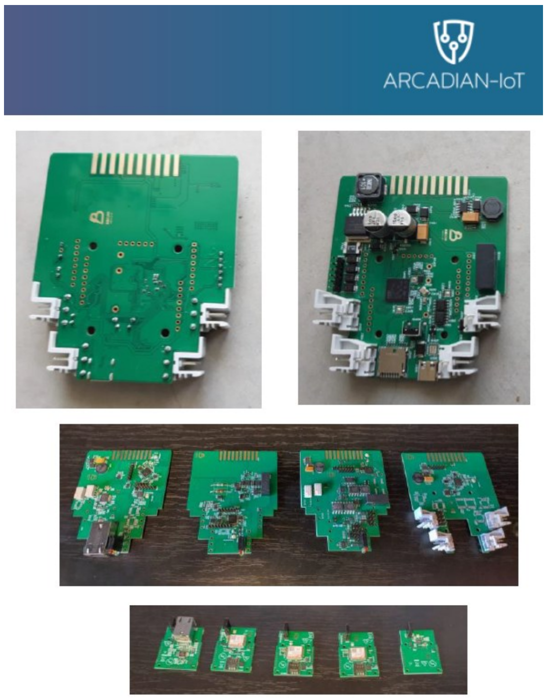


Figure 56 - Hardware prototypes

4.2.3 Design specification

4.2.3.1 Logical architecture view

4.2.3.1.1 Hardened encryption with crypto chip system

The architecture of the Hardened Encryption with the crypto chip system is presented in Figure 57.

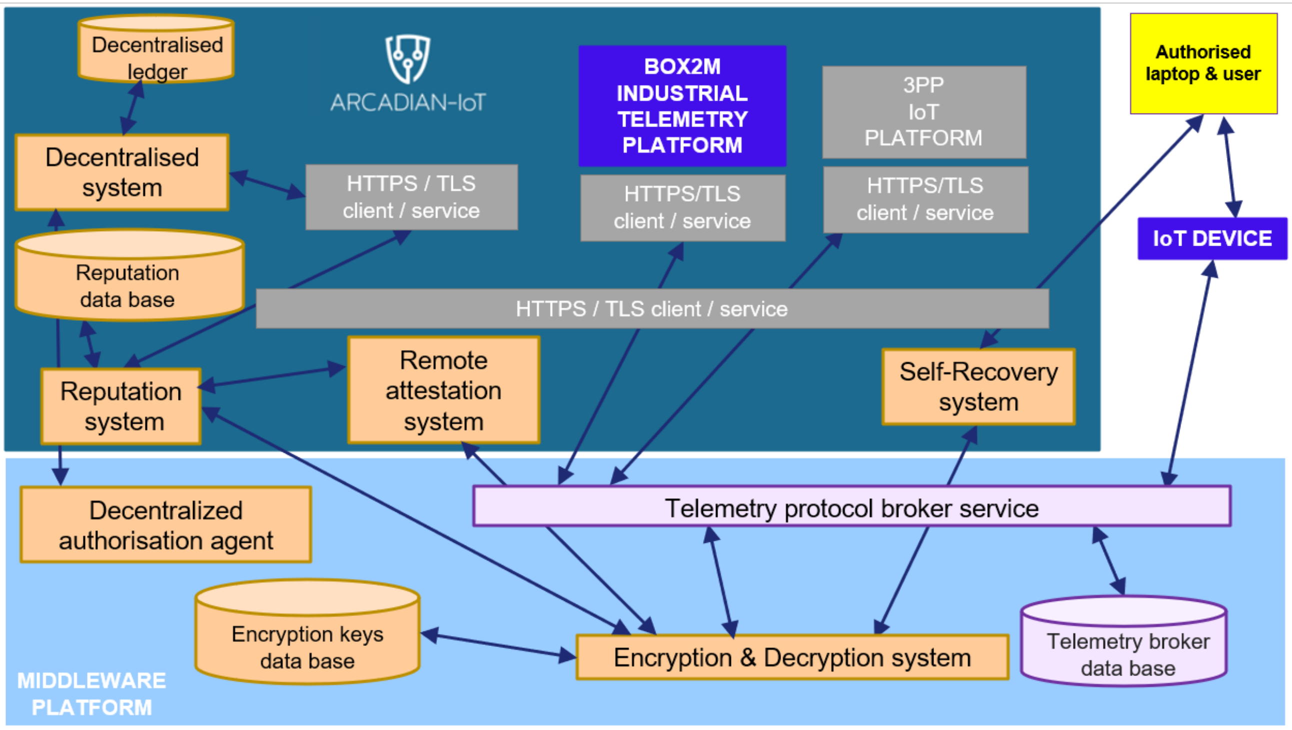

Decentralised ledger
ARCADIAN-IoT
BOX2M INDUSTRIAL TELEMETRY PLATFORM
3PP IoT PLATFORM
Authorised laptop & user
Decentralised system
HTTPS / TLS client / service
HTTPS/TLS client / service
HTTPS/TLS client / service
IoT DEVICE
Reputation data base
HTTPS / TLS client / service
Reputation system
Remote attestation system
Self-Recovery system
Decentralized authorisation agent
Telemetry protocol broker service
Encryption keys data base
Encryption & Decryption system
Telemetry broker data base
MIDDLEWARE PLATFORM

Figure 57 - Hardened Encryption (with crypto chip) system architecture

4.2.3.2 Sub-use cases

4.2.3.3 Sequence diagrams

The sequence diagram in Figure 58 and Figure 59 explains how the different subcomponents of the crypto chip-enabled Hardened Encryption interact during the onboarding and during traffic exchange.

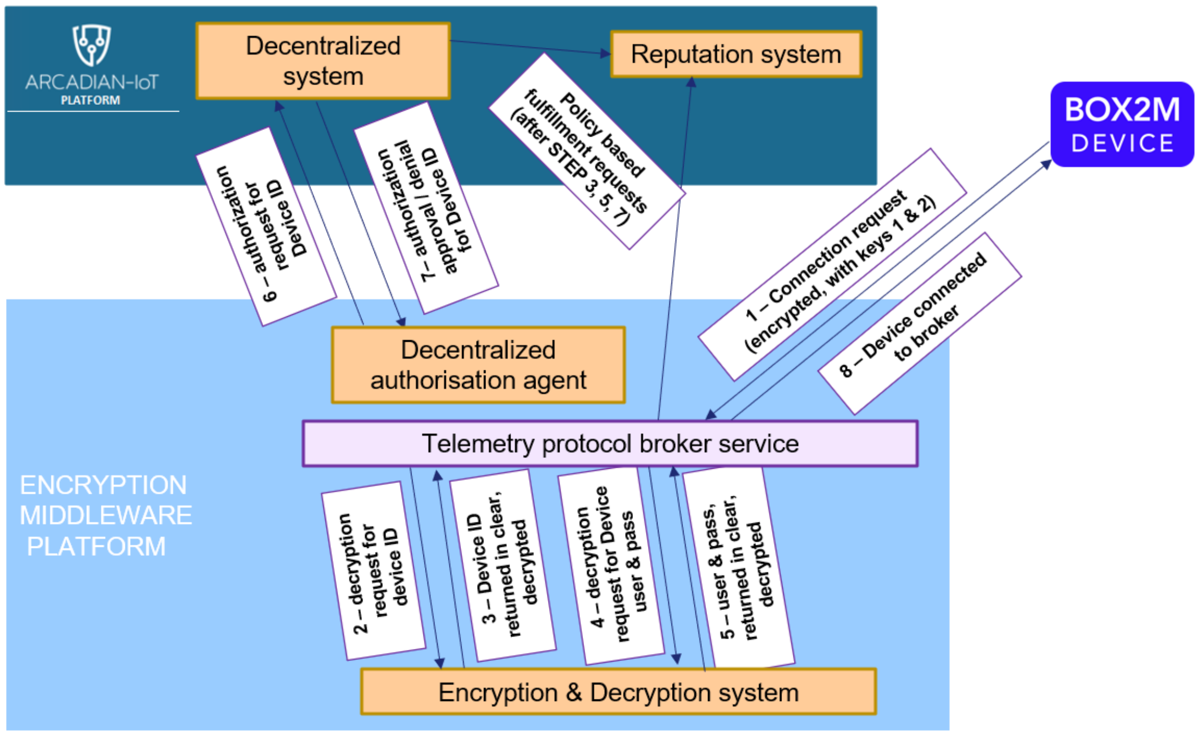

ARCADIAN-IoT PLATFORM
Decentralized system
Reputation system
BOX2M DEVICE
6 – authorization request for Device ID
7 – authorization approval / denial for Device ID
Policy based fulfillment requests (after STEP 3, 5, 7)
1 – Connection request (encrypted, with keys 1 & 2)
8 – Device connected to broker
Decentralized authorisation agent
Telemetry protocol broker service
ENCRYPTION MIDDLEWARE PLATFORM
2 – decryption request for device ID
3 – Device ID returned in clear, decrypted
4 – decryption request for Device user & pass
5 – user & pass, returned in clear, decrypted
Encryption & Decryption system

Figure 58 - Device onboarding

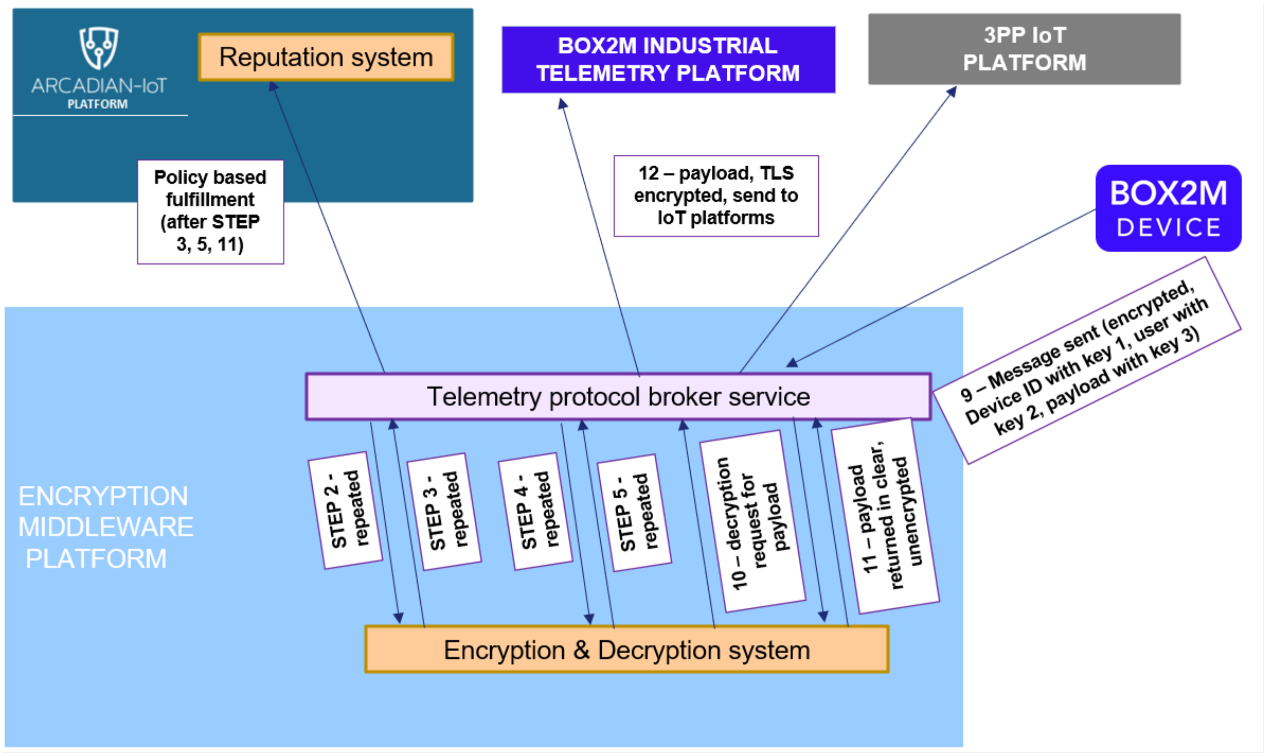

ARCADIAN-IoT PLATFORM
Reputation system
Policy based fulfillment (after STEP 3, 5, 11)
BOX2M INDUSTRIAL TELEMETRY PLATFORM
3PP IoT PLATFORM
12 – payload, TLS encrypted, send to IoT platforms
BOX2M DEVICE
9 – Message sent (encrypted, Device ID with key 1, user with key 2, payload with key 3)
Telemetry protocol broker service
ENCRYPTION MIDDLEWARE PLATFORM
STEP 2 - repeated
STEP 3 - repeated
STEP 4 - repeated
STEP 5 - repeated
10 – decryption request for payload
11 – payload returned in clear, unencrypted
Encryption & Decryption system

Figure 59 - Device traffic management

4.2.3.4 Interface description

A dedicated API orchestrator (part of the Middleware) was designed and can set up secure TLS-based communications with external systems. The API orchestrator enables rapid generation of multiple API channels (based on different criteria) which can operate simultaneously for the different integration purposes. This was used for interfacing with 3PP IoT platforms and with Remote Attestation System.

In parallel with the API, Middleware has designed and deployed a RabbitMQ interface, used for integration with all security systems.

Moreover, the IoT Middleware receives HTTP Signature for secure registration of constrained IoT devices and implements DIDs on behalf of IoT devices for secure authentication.

Partner	Definition	Channel	Exchange
IPN	Remote Attestation	API (Direct Code Integration)	Attestation requests from Verifier Attestation response (collected Evidence) - device attestation certificate
IPN	Device Behaviour Monitoring	RabbitMQ (Direct Code Integration)	Devices behaviour events
ATOS	Decentralized Identifier	RabbitMQ (Direct Code Integration)	Published Hardened Encryption Keys (DIDs)
MARTEL	Self-aware Data privacy	API & RabbitMQ	Enforcing privacy policies by encrypting the (device sensor) data

			in a database (of a 3PP IoT platform)
UC	Reputation System	RabbitMQ (Direct Code Integration)	Devices security events – correlated with communication stage and interpreted via a device reputation policy by Reputation System

4.2.3.5 Technical solution

4.2.3.5.1 Deployment architecture view

While the deployment view of the HE subcomponent for the ABE encryption and xSIM signatures has been shown in Figure 60, we report in Figure 60 how the crypto chip system deployment is designed in the Domain B.

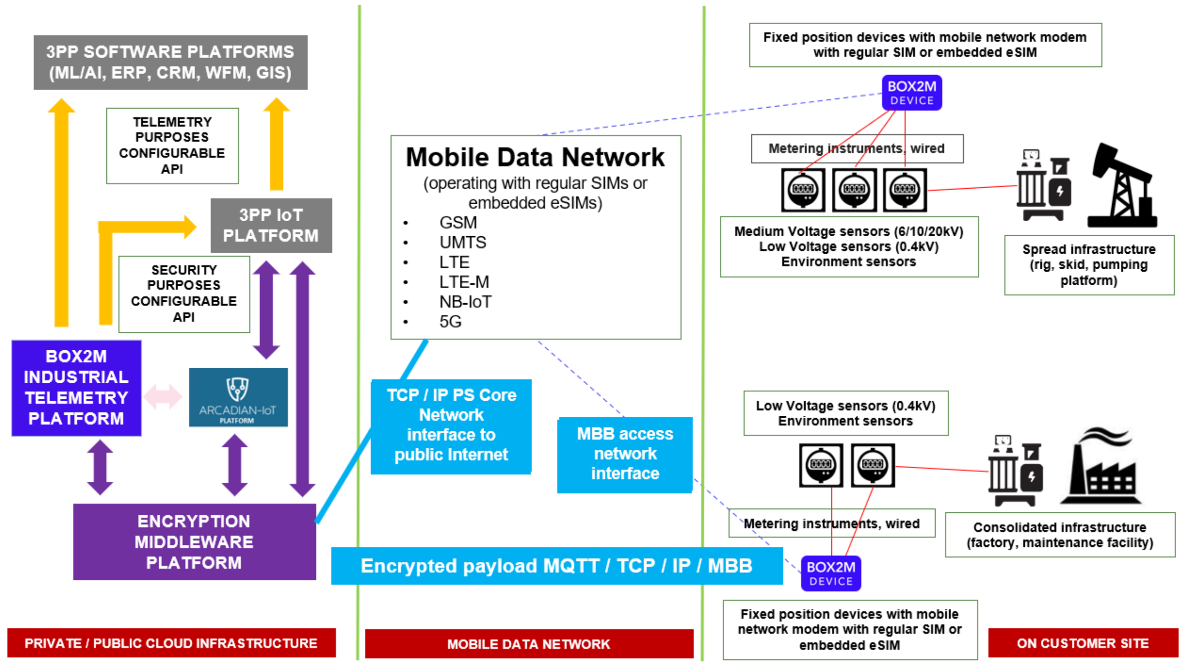
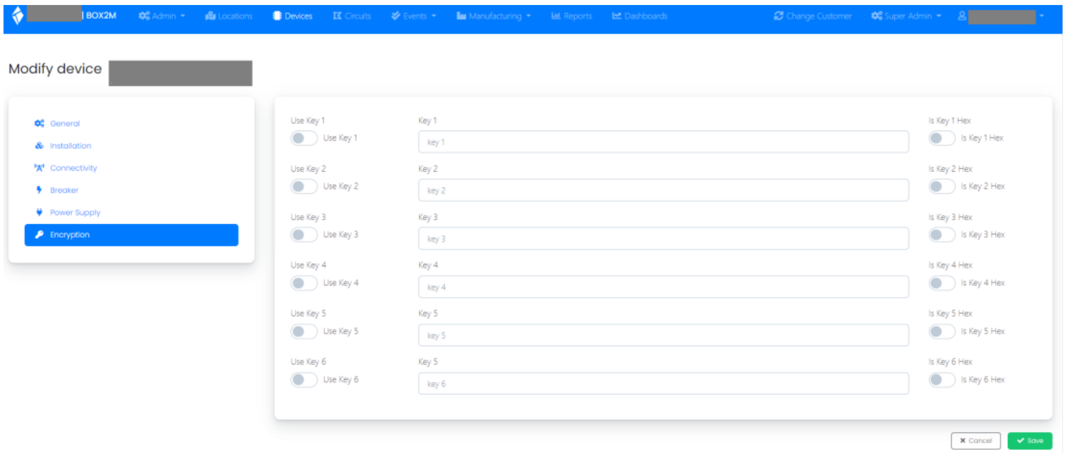

3PP SOFTWARE PLATFORMS (ML/AI, ERP, CRM, WFM, GIS)
TELEMETRY PURPOSES CONFIGURABLE API
3PP IoT PLATFORM
SECURITY PURPOSES CONFIGURABLE API
BOX2M INDUSTRIAL TELEMETRY PLATFORM
ARCADIAN-IoT PLATFORM
ENCRYPTION MIDDLEWARE PLATFORM
Mobile Data Network
(operating with regular SIMs or embedded eSIMs)
GSM
UMTS
LTE
LTE-M
NB-IoT
5G
TCP / IP PS Core Network interface to public Internet
MBB access network interface
Encrypted payload MQTT / TCP / IP / MBB
Fixed position devices with mobile network modem with regular SIM or embedded eSIM
BOX2M DEVICE
Metering instruments, wired
Medium Voltage sensors (6/10/20kV)
Low Voltage sensors (0.4kV)
Environment sensors
Spread infrastructure (rig, skid, pumping platform)
Low Voltage sensors (0.4kV)
Environment sensors
Metering instruments, wired
Consolidated infrastructure (factory, maintenance facility)
BOX2M DEVICE
Fixed position devices with mobile network modem with regular SIM or embedded eSIM
PRIVATE / PUBLIC CLOUD INFRASTRUCTURE
MOBILE DATA NETWORK
ON CUSTOMER SITE

Figure 60 - Grid domain implementation of crypto chip system

4.2.3.5.2 Frontend design

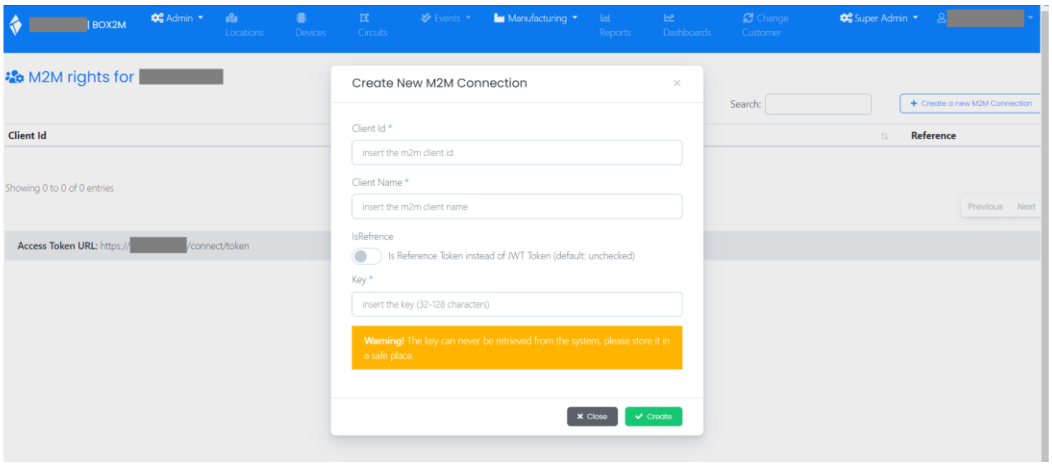
The crypto chip system includes a frontend for the key middleware and key-set up management, see Figure 61 and Figure 62.



Modify device [redacted]

Use Key	Key	Is Key Hex
Use Key 1: Use Key 1	Key 1: key 1	Is Key 1 Hex: Is Key 1 Hex
Use Key 2: Use Key 2	Key 2: key 2	Is Key 2 Hex: Is Key 2 Hex
Use Key 3: Use Key 3	Key 3: key 3	Is Key 3 Hex: Is Key 3 Hex
Use Key 4: Use Key 4	Key 4: key 4	Is Key 4 Hex: Is Key 4 Hex
Use Key 5: Use Key 5	Key 5: key 5	Is Key 5 Hex: Is Key 5 Hex
Use Key 6: Use Key 6	Key 5: key 6	Is Key 6 Hex: Is Key 6 Hex

Figure 61 - Crypto chip system - keys set-up (synced with device firmware provisioning)



M2M rights for [redacted]

Search:

Client Id	Reference

Showing 0 to 0 of 0 entries

Access Token URL: https://[redacted]/connect/token

Create New M2M Connection

Client Id *
insert the m2m client id

Client Name *
insert the m2m client name

IsRefrence
Is Reference Token instead of JWT Token (default: unchecked)

Key *
insert the key (32-128 characters)

Warning! The key can never be retrieved from the system, please store it in a safe place.

Figure 62 - Crypto chip system - middleware API set-up

4.2.3.5.3 Security aspects

The main security aspect relevant for the crypto chip system is the requirement to maintain the whole content secured end to end, from sensors to IoT platform and vice versa, despite using and relaying on 2 different security technologies (symmetrical and asymmetrical; by certificates – for TLS, and by dedicated keys – for crypto chip; by assembling a system from 2 different components – a firmware running into an MCU, mastering other sub firmwares, and a cloud Docker application).

A stress aspect for this end-to-end architecture is the data integrity, time and completeness aspects derived from encryption and decryption process.

4.2.4 Evaluation and results

In the case of the **Hardened Encryption using cryptochip system**, there were performed and demonstrated (see Figure 55) successfully next trial operations:

- 180+ hardware tests for the whole series of board types
- Device firmware provisioning with multiple sensors types
- Device firmware provisioning with multiple telecommunication types, engaged one by one mode – depending which board is plugged in

- xSIM provisioning on mobile broad band boards and testing, based on the board pluged in and engaged
- Regular SIM (multiple operators, including 2 MVNO) provisioning on mobile broad band boards and testing, based on the board plug in and engaged
- Failover telecommunication scenarios for dual telecommunication board (ETH and WiFi)
- Multiple devices provisioned into middleware (ID, user, password, keys for each stage)
- Multiple devices provisioned with correspondent keys and crypto chip firmware configured correspondently
- Encrypted and unencrypted authorisation scenarios
- Encryption and decryption authorisation scenarios with wrong key, in both ends (device and middleware), one by one and simultaneously
- Encryption and decryption authorisation scenarios with a valid key, but from different stage, in both ends (device and middleware), one by one and simultaneously
- Encryption and decryption traffic scenarios with wrong key, in both ends (device and middleware), one by one and simultaneously
- Encryption and decryption traffic scenarios with a valid key, but from different stage, in both ends (device and middleware), one by one and simultaneously
- False device attempt into middleware
- False middleware attempt into device
- Abnormal device operations (sudden reboot, repetitive reboot, telecommunication carrier interruption)
- Big data traffic stressing the middleware

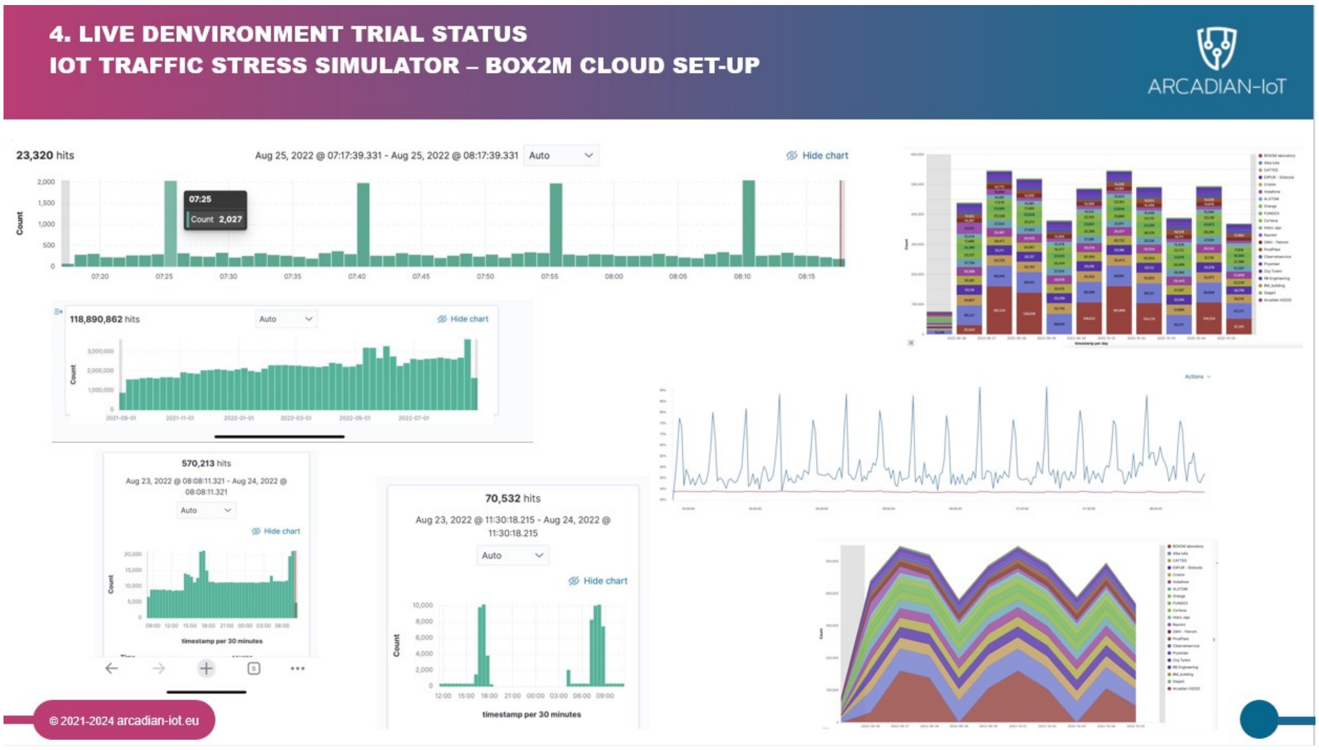

4. LIVE DENVIRONMENT TRIAL STATUS
IOT TRAFFIC STRESS SIMULATOR – BOX2M CLOUD SET-UP
ARCADIAN-IoT
23,320 hits
Aug 25, 2022 @ 07:17:39.331 - Aug 25, 2022 @ 08:17:39.331
Auto
Hide chart
07:25
Count 2,027
118,890,862 hits
570,213 hits
Aug 23, 2022 @ 08:08:11.321 - Aug 24, 2022 @ 08:08:11.321
70,532 hits
Aug 23, 2022 @ 11:30:18.215 - Aug 24, 2022 @ 11:30:18.215
timestamp per 30 minutes
Count
© 2021-2024 arcadian-iot.eu

Figure 63 - IoT stress simulator

4.2.5 Future work

- For crypto chip system, there are three scheduled tasks: (i) device hardware certification (for TRL6 readiness, estimated for end of Q1 2024), (ii) cyber security audit of Hardened Encryption System (wholistic and specific penetration tests with a Gov certified entity, across all system interfaces and ports, in scope of formalizing the Root of Trust) and (iii) maintaining the testing lab up & running with physical prototypes and virtual devices (with support of IoT devices simulator) performing repeatedly all encryption stages (registration, authentication, traffic), with correct and failed scenarios, to stress the whole system and interfaces with the other integrated ones. These results will strengthen the BOX2M Engineering position on commercial market and accelerate transition to TRL 9 (estimated for Q4 2024).

4.3 Permissioned Blockchain

4.3.1 Overview

4.3.1.1 Description

The Permissioned Blockchain is a common horizontal component in the ARCADIAN-IoT framework serving both Horizontal and Vertical Plane components such as Decentralized Identifiers and Hardened Encryption, which make use of its immutable auditability and traceability properties.

Given the sensitive nature of data that will be shared in the network, ARCADIAN-IoT the project initially requested to use a private Permissioned Blockchain approach for all use cases, but this will again be re-evaluated upon deeper analysis of each use case. Permissioned Blockchains place restrictions on who is allowed to participate in the network and in what read or write transactions. It can have several conditional access features for users to obtain permission to operate at given levels.

In order to interact with the blockchain, software clients typically run their own node, automatically updating the common state internally and then notifying the rest of nodes. Importantly the Permissioned Blockchain has much higher throughput and aimed more at enterprise solutions whereas the public permissionless blockchains are much slower due to their global participation and complex consensus proofs.

4.3.1.2 Requirements

The high-level requirement that has been previously defined and provided in deliverable D2.4 [19] has been updated as shown below.

Requirement 4.2.1 – Provide a Permissioned Blockchain:

- To provide a Permissioned Blockchain to anchor the trust for Decentralized identifiers.
- To provide a Permissioned Blockchain to publish information to be shared in a trusted and immutable fashion with different actors in the ecosystem, e.g., reputation scores for things, persons and services.
- Users have the right to delete all personal information published on them stored on ARCADIAN-IoT components.
- To be able to support the deletion of personal and device information it is needed to make use of off-chain databases that have their trust anchored in the blockchain.

4.3.1.3 Objectives and KPIs

The primary objective is to deploy an open-source Permissioned Blockchain network to support the other components in the ARCADIAN-IoT architecture that will benefit from its decentralized immutable auditability and traceability properties. The blockchain component will be evaluated against the following KPIs in all use case domains:

KPI scope	
Using the permissioned blockchain to publish trusted information to third parties in the ARCADIAN-IoT framework.	
Measurable Indicator	
Number of ARCADIAN-IoT services using permissioned blockchain	
Target value	***Achieved value***
3	4

KPI scope	
Deployment of permissioned blockchain in IoT environments	
Measurable Indicator	
Number of peer nodes deployed.	
Target value	***Achieved value***
3	2[32]

KPI scope	
Train partners to deploy a blockchain network	
Measurable Indicator	
Facilitate deployment of blockchain technologies by non-cyber security experts in cyber security training sessions with	
Target value	***Achieved value***
20	38

4.3.2 Technology research

4.3.2.1 Background

ATOS does not have any background assets to build upon in providing the Permissioned Blockchain to meet the needs of ARCADIAN-IoT. Therefore, only the current State-Of-The-Art in Permissioned Blockchain technology is described in this section.

Hyperledger Fabric [20]

Fabric is an open-source enterprise-grade permissioned Distributed Ledger Technology (DLT) platform established under the Linux Foundation and currently has reached a development community of over 35 organizations and nearly 200 developers. It is designed for the use in

[32] It is planned to reach the KPI target for the piloting to be reported in D5.5.

enterprise contexts, which delivers some key differentiating capabilities over other popular distributed ledger or blockchain platforms, and has a highly modular and configurable architecture, enabling innovation, versatility, and optimization for a broad range of industry use cases (including banking, finance, insurance, healthcare, human resources, supply chain and even digital music delivery).

It is the first distributed ledger platform to support smart contracts authored in general-purpose programming languages such as Java, Go and Node.js, rather than constrained domain-specific languages, so organisations benefit from existing competence.

The platform is also permissioned, meaning that, unlike with a public permissionless network, the participants are known to each other, rather than anonymous and therefore fully untrusted. This means that while the participants may not fully trust one another (they may, for example, be competitors in the same industry), a network can be operated under a governance model that is built off of what trust does exist between participants.

Key features of Hyperledger fabric are described in [29] and listed as follows:

- Rich queries over an immutable distributed ledger
- Support permissioned access to on-chain data on a need-to-know basis
- Support private data collections to protect personal and sensitive data off-chain
- Modular architecture supporting plug-in components
- Permissioned Membership Service
- Performance, scalability, and levels of trust
- Protection of digital keys and sensitive data.

Hyperledger Besu[33]

Besu is an open-source Ethereum client developed under the Apache 2.0 license and written in Java. It is designed to run on the Ethereum public network, but can also run on private permissioned networks, as well as test networks such as Rinkeby, Ropsten, and Görli. Hyperledger Besu supports several proof-of-authority algorithms (including QBFT, IBFT 2.0, and Clique) and proof-of-work (Ethash) consensus mechanisms.

Besu blockchain supports development of enterprise applications in Solidity to deliver secure, high-performance transaction processing in a private permissioned network.

Besu includes a command line interface and JSON-RPC API for running, maintaining, debugging, and monitoring nodes in an Ethereum network. You can use the API via RPC over HTTP or via WebSockets. The API supports typical Ethereum functionalities such as (i) Ether mining, (ii) Smart contract development, and (iii) Decentralized application (dApp) development.

Quorum[34]

Quorum provides a private permissioned network based on Ethereum for running smart contracts aimed at the banking and finance sector. Quorum strengths are high speed, scalability, and fast processing of private transactions. Its high speed is due to its simple consensus mechanisms (RAFT, IBFT Clique PoA, QBFT), and because of being an extension of the Ethereum platform; thus, most of the updates on Ethereum can be easily integrated in Quorum.

[33] https://besu.hyperledger.org/
[34] https://github.com/ConsenSys/quorum

Quorum uses Solidity for smart contract developments; according to the DLT platforms comparison provided by Alastria,[35] Quorum is apparently less community active in recent years while there is heavy competition in the Enterprise Ethereum area and platforms such as BlockApps and Hyperledger Besu.

Quorum networks can be used for a wide variety of use cases. In general, it seems to be the norm for banking and exchange-based applications, due to the benefits provided in terms of privacy brought to running processes, such as payments or post trade settlements.

IOTA Tangle[36]

IOTA Tangle is a DLT platform focused on IoT and provides a public permissionless network as well as, more recently, a private permissioned capability. IOTA nodes are thin, making possible to have an IOTA node running directly on a sensor. IOTA uses a persistence layer called IOTA Tangle which is a type of Directed Acyclic Graph with certain minor changes and a rival technology to blockchain. Whereas public blockchains rely on miners to select and aggregate the transactions with the highest fees into a sequential chain of blocks, the Tangle can process transactions in parallel, scaling with growing activity in the network.

4.3.2.2 Research findings and achievements

The primary characteristics that blockchain technology facilitates to components of the ACADIAN-IoT framework are as follows:

- ***Decentralized:*** Blockchain provides a decentralised peer-to-peer network of nodes that maintain an immutable record on a fault-tolerant ledger through consensus mechanisms. The decentralization means that all peer nodes have a copy of the ledger and access the same information so the greater number of nodes in the network the greater the fault tolerance. The consensus mechanisms facilitate that there is no central authority providing a trust anchor which facilitates the trust in the integrity of the data stored in the distributed ledger.
- **Transparent:** The decentralized network also means that any participant in the blockchain network can perform transactions with any other participant in a transparent manner. The transparency is achieved by the fact that all data on the ledger (on-chain) is available to all participants who have access to the ledger.
- **Private:** Privacy can be maintained by different means for example by providing a Permissioned Blockchain and also creating private blockchain subnetworks or channels where only those participants have access or by only storing hashes of private data on the ledger to be used to later check the integrity of private data stored (off-chain).
- **Immutable:** The intrinsic design of the recording blocks of data on the ledger provides for an incorruptible storage of data assuring its integrity from that point. This integrity of the blockchain also means that all transactions that enter data into the ledger are recorded and cannot be removed which has to be taken into consideration for storing of personal data due to the GDPR article on the “right to be forgotten.”

The ARCADIAN-IoT framework will be supported with a blockchain network of at least three peer nodes, as per the non-normative example below, with its component elements also described.

[35] https://alastria-es.medium.com/comparison-of-dlt-platforms-be84950d339d
[36] https://www.iota.org/

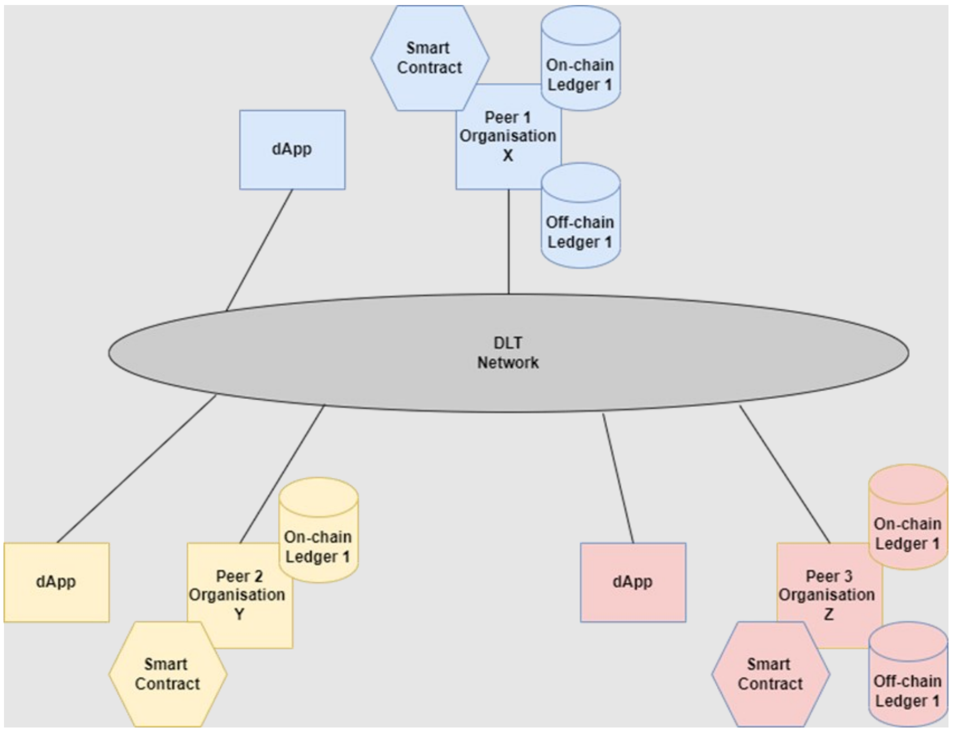


Figure 64 - Example of a blockchain network with three peer nodes.

The blockchain network shown above is composed of three organisation deployments with the different colours representing each deployment. The functions offered by each element are described as follows:

- **Smart Contract:** This is essentially the backend business logic of a decentralized blockchain application running on the peer nodes. A smart contract functions as a trusted distributed application that gains its security and trust from the blockchain and the underlying consensus among the peers.
- **dApp:** This is the decentralized client application that requests the Smart Contract on the blockchain peers to carry out a transaction on a business object / asset and publish the result to the ledger.
- **Peer:** This implements the consensus protocol between the peers in the network for reaching consensus in publishing data to the ledger.
- **On-chain Ledger:** This is the immutable writing transaction on a ledger based on blocks of data protected by crypto proofs on previous block written on the ledger. Data written on the on-chain ledger cannot be deleted as deleting any block would violate the trust in the ledger.
- **Off-chain Ledger:** This is a data store where the integrity of the entries is provided by cryptographic hash stored on the blockchain. This is an important function for processing personal data and also data that needs to be shared only between a few private actors.

Note that the peer nodes will be hosted by trusted project partners who participate in the publishing of data on the blockchain.

The combination of blockchain characteristics, previously described, makes the blockchain useful for applications that would benefit from a decentralized trust model. Within ARCADIAN-IoT the primary use of the blockchain is to allow some trusted actors to write on the blockchain so as to publish data that is made available to its participants and shared with third parties which can then verify the integrity of the data against the blockchain. It is important to highlight that no personal data shall be stored on-chain.

The blockchain was primarily envisioned to support the publishing of trusted data objects for the following components of the ARCADIAN-IoT framework: (i) Decentralized Identifiers, (ii) Reputation System, and (iii) Hardened Encryption. It was later, added on in the final prototype the support for establishing a trusted register of Service Providers.

Since the data published on a blockchain is intrinsically dynamic and may also include personal information, it is normal practice that business objects/assets that need to be published should be stored off-chain while a crypto hash of the asset is instead stored on-chain.

Although a private Permissioned Blockchain might be preferred in some use cases, Decentralised Identifier methods for public entities (e.g., public persons, services, and things) are commonly deployed in publicly available blockchains. For this reason, ARCADIAN-IoT will deploy a Permissioned Blockchain network where only some participants will be able to write on it, but all members can read it; that is, both private and public Permissioned Blockchain networks with mechanisms of access control (stating which entities have restricted access) will be supported.

DLT Selection for ARCADIAN-IoT

The evaluation approach in deciding which DLT to employ in the ARCADIAN-IoT framework, based on the State-of-the-Art analysed in section 4.3.2.1, considered the following aspects:

- The support to different use cases (specifically ARCADIAN-IoT use cases and requirements)
- The running efficiency and optimal use of resources
- The ability to develop in commonly used development languages
- The open-source developer community
- The provided support documents and tutorials
- The off-the-shelf support of private data that needs to be kept off chain.

Upon analysis of the open source permissioned blockchains, and the identified use cases for its application, it was decided in the first instance to support ARCADIAN-IoT with Hyperledger Fabric. Hyperledger Fabric comes out very strong in all of the above mentioned aspects and in particular an efficient use of resources is achieved as most of the data can be stored off-chain, making use of its Private Data Collections while being notarized with on-chain hashes; also greater throughput and scalability are general characteristics of Permissioned Blockchains aimed at enterprise applications, due to only a few authorised organisations allowed to write to the blockchain, leading to more efficient consensus algorithms. Note that the methodology that can be used to evaluate Hyperledger Fabric smart contract transactions with benchmark figures is available on Hyperledger's GitHub.[37]

This selection process ensured that the best state-of-the-art permissioned blockchain platform for the needs of ARCADIAN-IoT was used for the development of smart contracts delivered in this component.

4.3.2.3 Produced resources

The primary resource of the ARCADIAN-IoT Permissioned Blockchain component is a smart contract publishing ARCADIAN-IoT data objects on top of a deployed Hyperledger Fabric network.

[37] https://hyperledger.github.io/caliper-benchmarks/fabric/performance/

4.3.3 Design specification

4.3.3.1 Sub-use cases

4.3.3.1.1 Publish ARCADIAN-IoT Data to Permissioned Blockchain

The use case below shows the ARCADIAN-IoT Systems of Reputation and Hardened Encryption publishing data objects to the Permissioned Blockchain. It is seen that only hashes of the ARCADIAN-IoT object are stored on-chain.

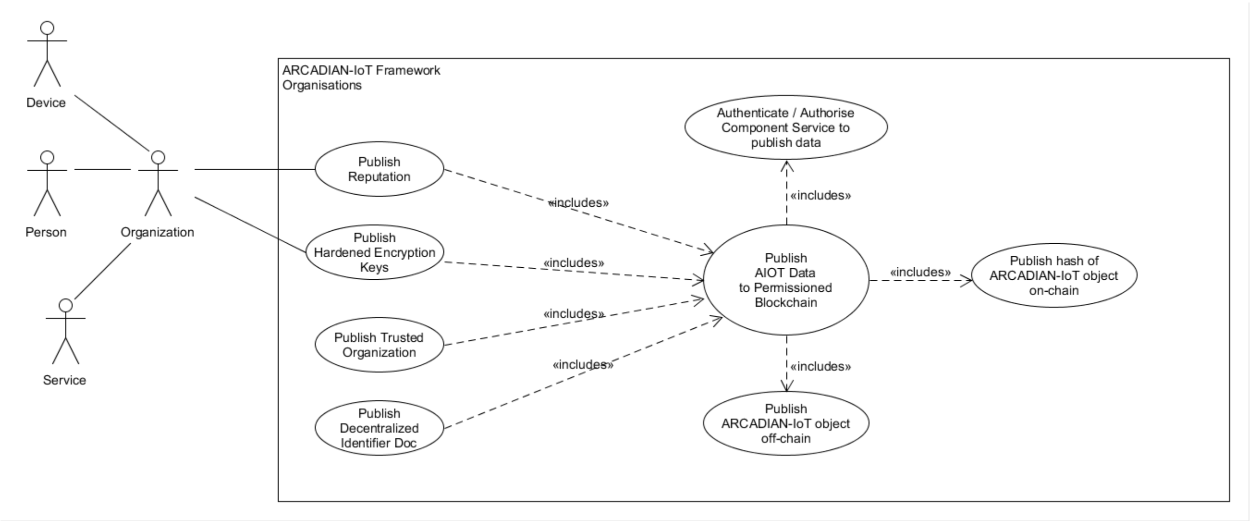

Device
Person
Organization
Service
ARCADIAN-IoT Framework Organisations
Publish Reputation
Publish Hardened Encryption Keys
Publish Trusted Organization
Publish Decentralized Identifier Doc
Authenticate / Authorise Component Service to publish data
Publish AIOT Data to Permissioned Blockchain
Publish hash of ARCADIAN-IoT object on-chain
Publish ARCADIAN-IoT object off-chain
«includes»

Figure 65 - Publish ARCADIAN-IoT Data to Permissioned Blockchain

It is noted that the component service publishing the data must be authenticated and authorised.

The organisation that carries out the publish transaction on-behalf of the component services must also have the correct permissions on the blockchain network.

4.3.3.1.2 Read Published ARCADIAN-IoT Data Objects

In this use case the previously published ARCADIAN-IoT Data is able to be read by the components systems that published the data. The retrieved data objects are validated against their on-chain hashes to make sure they have not been tampered with.

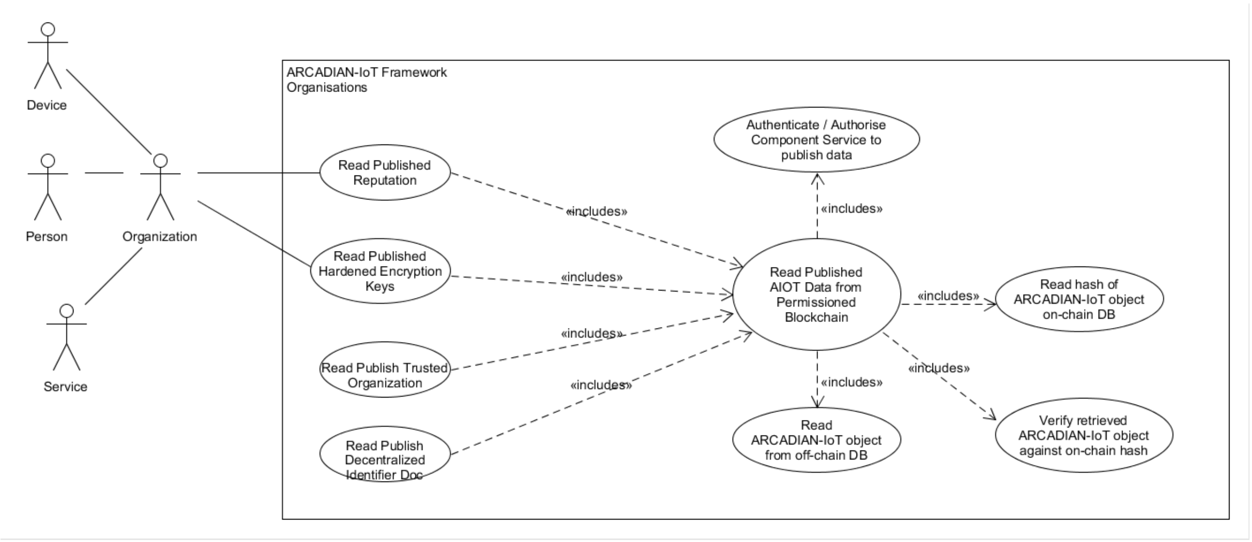

Device
Person
Organization
Service
ARCADIAN-IoT Framework Organisations
Read Published Reputation
Read Published Hardened Encryption Keys
Read Publish Trusted Organization
Read Publish Decentralized Identifier Doc
Authenticate / Authorise Component Service to publish data
Read Published AIOT Data from Permissioned Blockchain
Read hash of ARCADIAN-IoT object on-chain DB
Verify retrieved ARCADIAN-IoT object against on-chain hash
Read ARCADIAN-IoT object from off-chain DB
«includes»

Figure 66 - Read published ARCADIAN-IoT Data Objects

It is noted that the component service reading the published data must be authenticated and authorised.

The organisation that carries out transaction on-behalf of the component services to read published data must also have the correct permissions on the blockchain network.

4.3.3.2 Logical architecture view

4.3.3.2.1 Publish ARCADIAN-IoT Data Objects

The logical view of the Permissioned Blockchain node for publishing ARCADIAN-IoT Objects and Decentralized Identifiers from registered ARCADIAN-IoT components is shown below. The blank component is included to show that the architecture is extensible to support publishing of any data object from any ARCADIAN-IoT component.

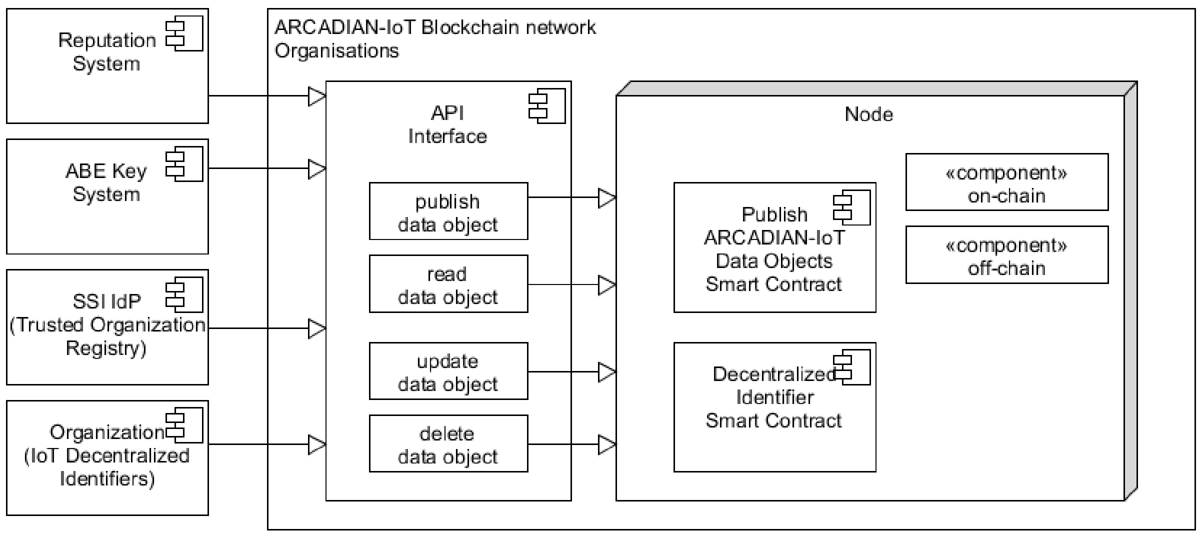

Reputation System
ABE Key System
SSI IdP (Trusted Organization Registry)
Organization (IoT Decentralized Identifiers)
ARCADIAN-IoT Blockchain network Organisations
API Interface
publish data object
read data object
update data object
delete data object
Node
Publish ARCADIAN-IoT Data Objects Smart Contract
Decentralized Identifier Smart Contract
«component» on-chain
«component» off-chain

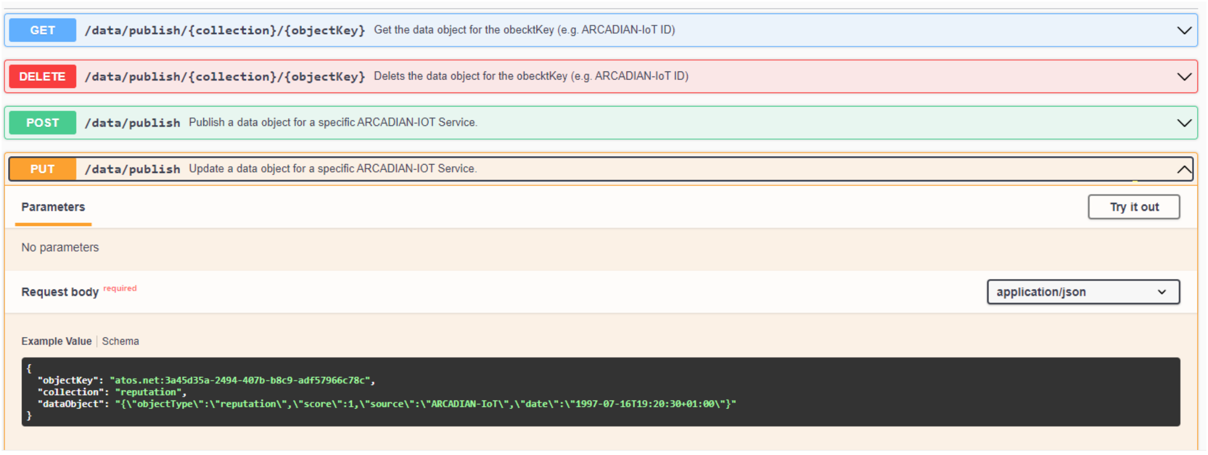
Figure 67 - Logical architecture for publishing ARCADIAN-IoT Objects to the Permissioned Blockchain

API Interface: This provides an interface for the trusted organizations with permission to write to the blockchain to publish data to the appropriate off-chain private data collections and associated on-chain hashes.

Node: The peer node runs the smart contract that is responsible for performing transactions to the on-chain and off-chain ledgers residing on the node as well as configuration and running of the consensus algorithm that controls the commits to the ledger.

4.3.3.3 Interface description

4.3.3.3.1 Publish ARCADIAN-IoT Data Objects


GET /data/publish/{collection}/{objectKey} Get the data object for the obecktKey (e.g. ARCADIAN-IoT ID)

DELETE /data/publish/{collection}/{objectKey} Delets the data object for the obecktKey (e.g. ARCADIAN-IoT ID)

POST /data/publish Publish a data object for a specific ARCADIAN-IOT Service.

PUT /data/publish Update a data object for a specific ARCADIAN-IOT Service.

Parameters

Try it out

No parameters

Request body required

application/json

Example Value | Schema

```
{
  "objectKey": "atos.net:3a45d35a-2494-407b-b8c9-adf57966c78c",
  "collection": "reputation",
  "dataObject": "{\"objectType\":\"reputation\",\"score\":1,\"source\":\"ARCADIAN-IoT\",\"date\":\"1997-07-16T19:20:30+01:00\"}"
}
```

Figure 68 - Publish ARCADIAN-IoT Data Objects Interface Description

4.3.3.3.2 *Publish DID transactions*

The Decentralized Identifier component in deliverable D4.3 provides a smart contract [29] to provide the DID transactions carrying out CRUD operations.

4.3.3.4 Technical solution

4.3.3.4.1 *Deployment architecture view*

4.3.3.4.2 *Publishing ARCADIAN-IoT Data deployment view*

The Permissioned Blockchain deployment architecture is based on Hyperledger Fabric 2.5 and three Peer Nodes will be deployed for supporting the publishing of ARCADIAN-IoT Data Objects and DIDs, as shown in the figure below.

There is one API Interface per ARCADIAN-IoT organisation that deploys a peer node with read / write permissions to the blockchain, as per the Membership Service Provider provided by Hyperledger Fabric based on X.509 certificates.

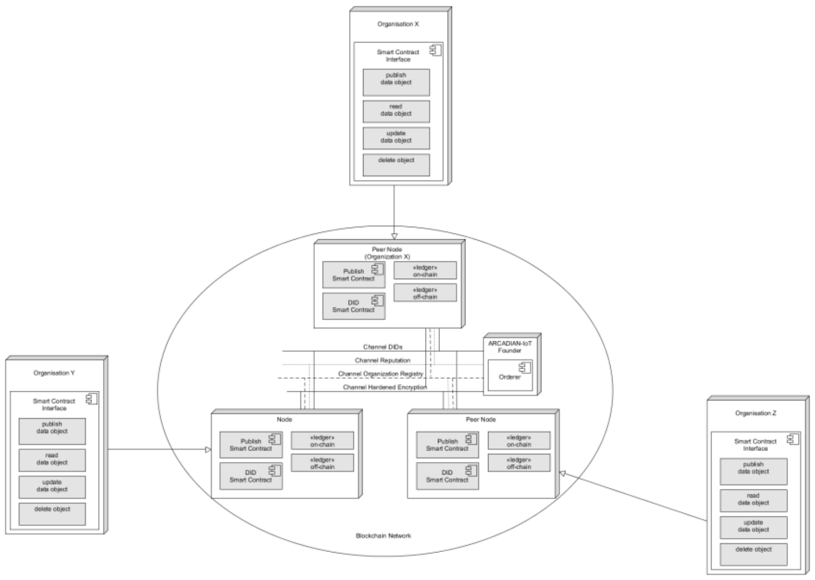


Figure 69 - Permissioned Blockchain for Publishing ARCADIAN-IoT Data deployment view

4.3.3.4.3 *Publishing Sidetree transactions deployment view*

Please refer to deliverable D4.3 - Vertical Planes [29] for the overall deployment view of the blockchain network supporting the Sidetree node.

4.3.3.4.4 *API specification*

The OpenAPI YAML specification is published on the ARCADIAN-IoT GitLab repository[38].

4.3.3.4.5 *Publishing ARCADIAN-IoT Data Smart Contract design*

Overview

The Smart Contract is designed to publish data objects in an opaque manner for ARCADIAN-IoT Identities for a type of object e.g. Reputation object, Attribute Based Encryption Key object.

The published data objects can be consumed by any component in the ARCADIAN-IoT Framework, and also any external organisation configured on the Blockchain Network channel.

Essentially, the permissioned blockchain provides ARCADIAN-IoT with a distributed and fault-tolerant trusted database for use by multiple instances of the ARCADIAN-IoT Framework and also with external parties such as Attribute Based Encryption Key Providers, external services and auditors etc.

All published data objects are published to an off-chain database and only the hashes of the published data are stored on-chain so to provide immutable integrity of the published data.

Channel´s Private Data Collections are able to be validated against the hash stored on the channels on-chain world state. Any private data can be deleted at any time as well as to be completely purged from transaction logs [17], only the hashed data on-chain will remain.

Channels

The organisations allowed to access the on-chain and off-chain world states of the blockchain are primarily controlled by being configured to a channel. In this way there will be multiple channels established in ARCADIAN-IoT deployment per business data object that will be published in that channel e.g. a Reputation Channel, ABE Key Channel.

Private Data Collections

Private Data Collections[39] will be used to publish data off-chain in a deterministic manner on a Hyperledger Fabric blockchain network[40], and each business object will have its own private data collection e.g. reputation collection.

The accesses to Private Data Collections are controlled by:

- The member organisations who were given permission to take part in that blockchain channel[41]
- A collection definition[42] JSON file to control member access in the channel to the collection

Private Data Collections only store the hash of their transaction on chain, so the state & all previous transactions can be checked against on-chain hash records.

[38] https://gitlab.com/arcadian_iot/blockchain/-/blob/main/interface/openapiPB.yml

[39] Private Data Collections

[40] Hyperledger Fabric blockchain network

[41] blockchain channel

[42] collection definition JSON

CouchDB is identified for deployment as the database as can be extended in the future so that more complex SQL type relational DB queries can be defined using indexes[43].

Data Model

For publishing Reputation data an off-chain Private Data Collection specific to the type of Reputation data object to be published is created, as follows:

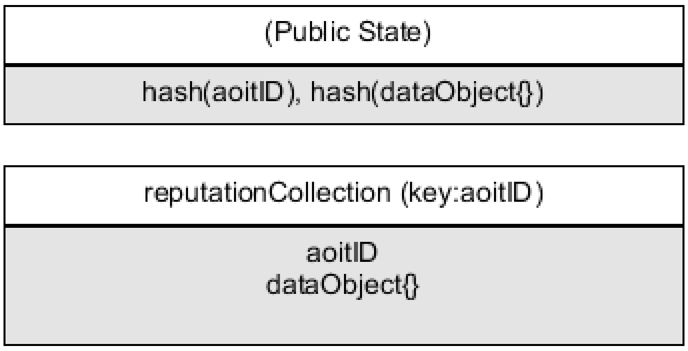

(Public State)
hash(aoitID), hash(dataObject{})
reputationCollection (key:aoitID)
aoitID
dataObject{}

Figure 70 - Reputation Private Data Collection

For publishing Hardened Encryption data then this follows the same approach to create an off-chain Private Data Collection specific to the type of encryption key data object to be published. For example, the Attribute Based Encryption Key object will implement the following Private Data Collection:

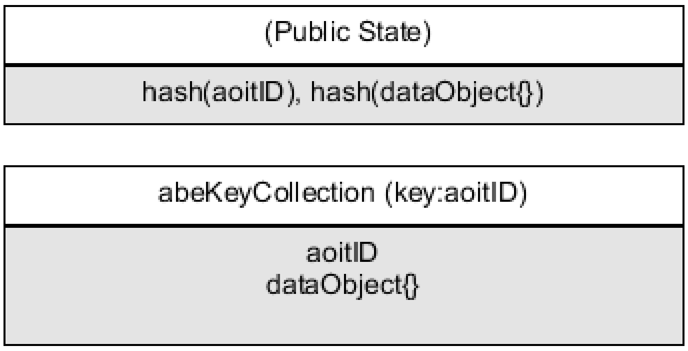

(Public State)
hash(aoitID), hash(dataObject{})
abeKeyCollection (key:aoitID)
aoitID
dataObject{}

Figure 71 - ABE Key Private Data Collection

ARCADIAN-IoT supports publishing trusted organization´s in the permissioned blockchain so that only trusted organizations can interact within the ARCADIAN-IoT ecosystem. The Private Data Collection is defined as follows with the index key being the Organization´s public DID:

[43] https://hyperledger-fabric.readthedocs.io/en/latest/couchdb_tutorial.html

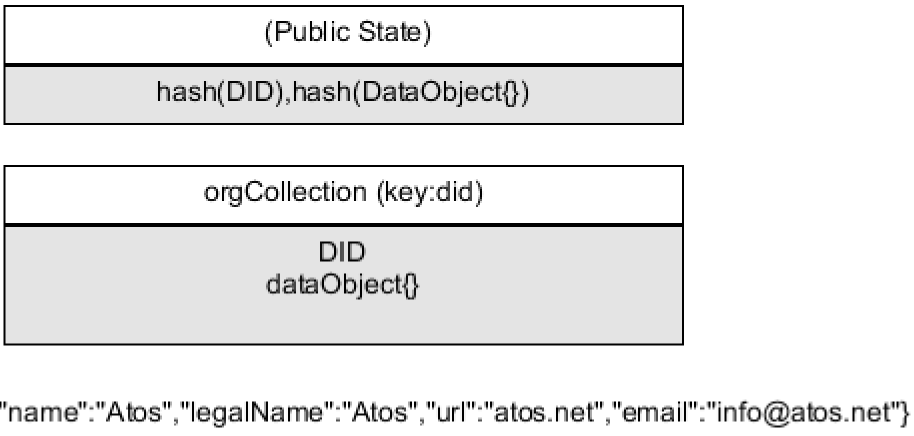

(Public State)
hash(DID),hash(DataObject{})
orgCollection (key:did)
DID
dataObject{}
dataObject{"name":"Atos","legalName":"Atos","url":"atos.net","email":"info@atos.net"}

Figure 72 - Organization Private Data Collection

ARCADIAN-IoT supports publishing Decentralized Identifiers (for use inside the ARCADIAN-IoT ecosystem) on the permissioned blockchain, so that only authorized organizations can publish and later control these DIDs. The Private Data Collection is defined as follows:

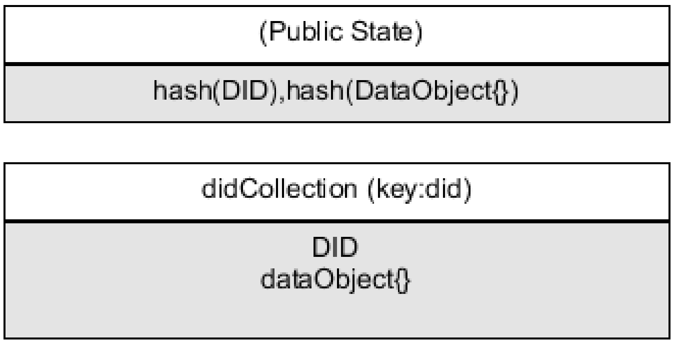


Figure 73 - Decentralized identifier Private Data Collection

Chaincode

The smart contract is deployed on peer nodes as chaincode and include the private collections definition file. The peer lifecycle chaincode subcommand allows administrators to use the Fabric chaincode lifecycle to package a chaincode, install it on your peers, approve a chaincode definition for your organization, and then commit the definition to a channel [15].

Transactions

To publish data to the off-chain Private Data Collection the Hyperledger Fabric Contract API is called as follows:

```
PutPrivateData(collection string, key string, value []byte)
```

The API calls from the Smart Contract GW Interface for Posting and updating a collections business data object will result in this PutPrivateData transaction.

To read data from the off-chain Private Data Collection the following Hyperledger Fabric Contract API is called:

```
GetPrivateData(collection string, key string) ([]byte, error)
```

The API call from the Smart Contract GW Interface to get a collection´s business data object will result in this GetPrivateData transaction.

To delete data from the off-chain Private Data Collection the following Hyperledger Fabric Contract API is called:

```
PurgePrivateData(collection string, key string)
```

The API call from the Smart Contract GW Interface to delete a collection´s business data object will result in this PurgePrivateData transaction. The transaction APIs are fully specified in Hyperledger Fabric [16].

Implementation Notes

The Smart Contract is written in JavaScript.

It is seen from the above transactions, that the data being committed to the Private Data Collection (key value pair) is a string stored as an array of bytes and the value actually being stored is a stringified JSON object. However, JSON is not a deterministic data format i.e. the order of elements can change, whilst still representing the same data semantically. The challenge, therefore, is to be able to generate a consistent set of JSON.

As the Smart Contract is written in Javascript, then to achieve consistent results, a deterministic version of JSON.stringify() is needed rather than the function JSON.stringify() which is commonly used when serialising a JSON object. In this way it is possible to get a consistent hash from stringified results. json-stringify-deterministic is the recommended library to do so and can also be used combined with sort-keys-recursive to attain alphabetic order too [18].

4.3.3.4.6 Security aspects

Only registered ARCADIAN-IoT component services will be able to request to create data object transactions to the smart contracts.

Any registered component or registered external party should be able to read it.

4.3.4 Evaluation and results

The smart contract implementation has been deployed on a Hyperledger Fabric test network in the pre-production environment and was tested with dummy data that the data objects were correctly published to the blockchain. It was subsequently made available to the piloting partner systems.

Ultimately, the evaluation of the Permissioned Blockchain in ARCADIAN-IoT framework and the smart contract(s) that are developed for its use will be obtained by the evaluation of the components that use it during piloting.

4.3.5 Future work

Based on the results of this project, Atos will consider incorporating the blockchain component as a Verifiable Data Registry into its identity portfolio, for supporting use cases with public Decentralized Identifiers.

5 CONCLUSIONS

The partners involved in producing this WP3 deliverable are IPN, ATOS, MAR, RISE, BOX2M, UWS, XLAB, and 1GLOBAL. The deliverable itself consists of this final report on the components in the Horizontal Planes of the ARCADIAN-IoT framework as well as the developed software for deployment in the piloting domains in WP5.

In summary, the ARCADIAN-IoT Horizontal components researched and developed as part of this deliverable are as follows:

- Self-aware Data Privacy
- Federated AI
- Network Flow Monitoring
- Behaviour Monitoring
- Cyber Threat Intelligence
- Network Self-protection
- IoT Device Self-protection
- Network Self-healing
- Hardened Encryption (SIM as RoT & Crytochip as RoT options)
- Permissioned Blockchain

This final iterative version of the deliverable (based on deliverable D3.2) reflects the Horizontal plan features enabled in the final P2 prototype, where the partners concluded the technological research of each component and updated their design for added features and improvements.

The description of the component prototypes and results achieved in this deliverable are input for the use cases in WP5 to finalise their domain integrations and piloting for the P2 prototype features.

REFERENCES

[1] De La Calleja, Jorge, and Olac Fuentes. "A Distance-Based Over-Sampling Method for Learning from Imbalanced Data Sets." In FLAIRS Conference, pp. 634-635. 2007.
[2] Mani, Inderjeet, and I. Zhang. "kNN approach to unbalanced data distributions: a case study involving information extraction." In Proceedings of workshop on learning from imbalanced datasets, vol. 126, pp. 1-7. ICML, 2003.
[3] Chawla, Nitesh V., Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. "SMOTE: synthetic minority over-sampling technique." Journal of artificial intelligence research 16 (2002): 321-357.
[4] Fernández, Alberto, Salvador Garcia, Francisco Herrera, and Nitesh V. Chawla. "SMOTE for learning from imbalanced data: progress and challenges, marking the 15-year anniversary." Journal of artificial intelligence research 61 (2018): 863-905.
[5] Alham Fikri Aji and Kenneth Heafield. 2017. Sparse Communication for Distributed Gradient Descent. In Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, pages 440–445, Copenhagen, Denmark. Association for Computational Linguistics.
[6] Tang, Zhenheng, Shaohuai Shi, and Xiaowen Chu. "Communication-efficient decentralized learning with sparsification and adaptive peer selection." 2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS). IEEE, 2020.
[7] Reisizadeh, Amirhossein, et al. "Fedpaq: A communication-efficient federated learning method with periodic averaging and quantization." International Conference on Artificial Intelligence and Statistics. PMLR, 2020.
[8] Zhou, Yuhao, Qing Ye, and Jiancheng Lv. "Communication-efficient federated learning with compensated overlap-fedavg." IEEE Transactions on Parallel and Distributed Systems 33.1 (2021): 192-205.
[9] [1] McMahan, Brendan, et al. "Communication-efficient learning of deep networks from decentralized data." Artificial intelligence and statistics. PMLR, 2017.
[10] Yin, Dong, et al. "Byzantine-robust distributed learning: Towards optimal statistical rates." International Conference on Machine Learning. PMLR, 2018.
[11] Blanchard, Peva, et al. "Machine learning with adversaries: Byzantine tolerant gradient descent." Advances in Neural Information Processing Systems 30 (2017).
[12] Wang, Han, Luis Muñoz-González, David Eklund, and Shahid Raza. "Non-IID data re-balancing at IoT edge with peer-to-peer federated learning for anomaly detection." In Proceedings of the 14th ACM Conference on Security and Privacy in Wireless and Mobile Networks, pp. 153-163. 2021.
[13] Koroniotis, Nickolaos, et al. "Towards the development of realistic botnet dataset in the internet of things for network forensic analytics: Bot-iot dataset." Future Generation Computer Systems 100 (2019): 779-796.
[14] Meidan, Yair, et al. "N-baiot—network-based detection of iot botnet attacks using deep autoencoders." IEEE Pervasive Computing 17.3 (2018): 12-22.
[15] Wang, Xinlei, et al. "Performance evaluation of attribute-based encryption: Toward data privacy in the IoT." *2014 IEEE International Conference on Communications (ICC)*. IEEE, 2014.
[16] Sahai, Amit, and Brent Waters. "Fuzzy identity-based encryption." In Annual international conference on the theory and applications of cryptographic techniques, pp. 457-473. Springer, Berlin, Heidelberg, 2005.
[17] Oberko, Prince Silas Kwesi, Victor-Hillary Kofi Setornyo Obeng, and Hu Xiong. "A survey on multi-authority and decentralized attribute-based encryption." *Journal of Ambient Intelligence and Humanized Computing* 13.1 (2022): 515-533.
[18] Perazzo, Pericle, et al. "Performance evaluation of attribute-based encryption on

constrained iot devices." Computer Communications 170 (2021): 151-163.

[19] Ambrosin, Moreno, Mauro Conti, and Tooska Dargahi. "On the feasibility of attribute-based encryption on smartphone devices." Proceedings of the 2015 Workshop on IoT challenges in Mobile and Industrial Systems. 2015.

[20] Agrawal, Shashank, and Melissa Chase. "FAME: fast attribute-based message encryption." *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. 2017.

[21] Lewko, Allison, and Brent Waters. "Decentralizing attribute-based encryption." *Annual international conference on the theory and applications of cryptographic techniques*. Springer, Berlin, Heidelberg, 2011.

[22] Lewko, Allison, and Brent Waters. "Decentralizing attribute-based encryption." *Annual international conference on the theory and applications of cryptographic techniques*. Springer, Berlin, Heidelberg, 2011.

[23] Agrawal, Shashank, and Melissa Chase. "FAME: fast attribute-based message encryption." *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. 2017.

[24] Damgård, I., Pastro, V., Smart, N., & Zakarias, S. (2012, August). Multiparty computation from somewhat homomorphic encryption. In *Annual Cryptology Conference* (pp. 643-662). Berlin, Heidelberg: Springer Berlin Heidelberg.

[25] Maurer, Ueli. "Secure multi-party computation made simple." *Discrete Applied Mathematics* 154.2 (2006): 370-381.

[26] Wang, Xinlei, et al. "Performance evaluation of attribute-based encryption: Toward data privacy in the IoT." *2014 IEEE International Conference on Communications (ICC)*. IEEE, 2014.

[27] Wang, Xinlei, et al. "Performance evaluation of attribute-based encryption: Toward data privacy in the IoT." *2014 IEEE International Conference on Communications (ICC)*. IEEE, 2014.

[28] Cocco, Sharon, and Gari Singh. "Top 6 technical advantages of Hyperledger Fabric for blockchain networks" (2018). Available from https://developer.ibm.com/articles/top-technical-advantages-of-hyperledger-fabric-for-blockchain-networks/

[29] ARCADIAN-IoT, "D4.3 - Vertical Planes – final version", 2024

[30] Private Data Collections tutorial. https://hyperledger-fabric.readthedocs.io/en/latest/private_data_tutorial.html , 2022

[31] Peer Lifecycle Chaincode, https://hyperledger-fabric.readthedocs.io/en/latest/commands/peerlifecycle.html#peer-lifecycle-chaincode-commit, 2022

[32] Hyperledger Fabric Chaincode APIs, https://hyperledger.github.io/fabric-chaincode-node/main/api/fabric-shim.ChaincodeStub.html#getPrivateData__anchor, 2022

[33] https://hyperledger-fabric.readthedocs.io/en/release-1.3/private-data/private-data.html, 2022

[34] https://hyperledger-fabric.readthedocs.io/en/release-2.4/chaincode4ade.html, 2022

[35] ARCADIAN-IoT, "D2.4 ARCADIAN-IoT framework requirements", 2021

[36] https://hyperledger-fabric.readthedocs.io/en/release-2.4/whatis.html, 2022

[37] J. Snoek, H. Larochelle, R. P. Adams, Practical bayesian optimization of machine learning algorithms, in: F. Pereira, C. J. C. Burges, L. Bottou, K. Q. Weinberger (Eds.), Advances in Neural Information Processing Systems, Vol. 25, Curran Associates, Inc., 2012.

[38] Borisaniya, B. and Patel, D. (2015) Evaluation of Modified Vector Space Representation Using ADFA-LD and ADFA-WD Datasets. Journal of Information Security, 6, 250-264

[39] Stephanie Forrest, Steven A. Hofmeyr, Anil Somayaji, and Thomas A. Longstaff. A sense of self for unix processes. In Proceedings of the 1996 IEEE Symposium on Security and Privacy, 1996

[40] W. Lee and S. Stolfo. Data mining approaches for intrusion detection. In Proceedings of the 7th USENIX Security Symposium, 1998.

[41] P. Helman and J. Bhangoo. A statistically based system for prioritizing information exploration under uncertainty. Trans. Sys. Man Cyber, 1997.
[42] Yihua Liao and Rao Vemuri. Using text categorization techniques for intrusion detection, 2002.
[43] F. T. Liu, K. M. Ting and Z. -H. Zhou, "Isolation Forest," 2008 Eighth IEEE International Conference on Data Mining, Pisa, Italy, 2008, pp. 413-422, doi: 10.1109/ICDM.2008.17.
[44] M. Wang and K. Zheng and Y. Yang and X. Wang, An Explainable Machine Learning Framework for internet of things Detection Systems, 2020
[45] Patil, Shruti and Varadarajan, Vijayakumar and Mazhar, Siddiqui Mohd and Sahibzada, Abdulwodood and Ahmed, Nihal and Sinha, Onkar and Kumar, Satish and Shaw, Kailash and Kotecha, Ketan; Explainable Artificial Intelligence for Intrusion Detection System; 2022
[46] Marino, Daniel L. and Wickramasinghe, Chathurika S. and Manic, Milos; An Adversarial Approach for Explainable AI in Intrusion Detection Systems; 2018
[47] Shraddha Mane and Dattaraj Rao, Explaining Network Intrusion Detection System Using Explainable AI Framework, 2021
[48] https://research.unsw.edu.au/projects/adfa-ids-datasets
[49] https://github.com/torvalds/linux/tree/master/tools/perf
[50] https://gitlab.com/arcadian_iot/device_behaviour_monitoring
[51] FK Došilović, M Brčić, and N Hlupić, Explainable artificial intelligence: A survey, 2018
[52] https://www.appsealing.com/owasp-iot-top-10/
[53] Relationships to Other Concepts - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/figure/The-MAPE-K-architecture-for-autonomic-managers-inspired-by-190-The-monitor-and_fig2_305706253 [accessed 23 Nov 2022]
[54] https://gitlab.com/arcadian_iot/device_self_protection
[55] https://github.com/open-policy-agent/opa/releases