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

ARCADIAN-IoT

Autonomous Trust, Security and Privacy Management Framework for IoT

D3.2: Horizontal Planes - second version

Revision: v1.0

Work package	3
Task	Tasks 3.1, 3.2, 3.3, 3.4, and 3.5
Due date	31/12/2022
Submission date	04/01/2023
Deliverable lead	XLAB
Version	1.0
Partner(s) / Author(s)	RISE: Alfonso Iacovazzi, Han Wang, İsmail Bütün IPN: João Rainho, Paulo Silva, Vitalina Holubenko

	UWS: Jose M. Alcaraz Calero, Qi Wang, Antonio Matencio Escolar, Ignacio Sánchez Navarro, Pablo Benlloch Caballero TRU: João Casal, Ivo Vilas Boas, Tomás Silva XLAB: Tilen Marc, Nejc Bat MARTEL: Giacomo Inches, Gabriele Cerfoglio, Federico Facca BOX2M: Alexandru Gliga, Ovidiu Diaconescu, Marian Macoveanu ATOS: Ross Little

Abstract

This public technical report constitutes the deliverable D3.2 of ARCADIAN-IoT, a Horizon2020 project with **the grant agreement number 101020259**, under the topic **SU-DS02-2020**. D3.2 is the second of a series of three deliverables planned for reporting the findings, achievements, and outcomes resulted from WP3 research activity.

The material in this document presents the main outcome of all tasks in WP3 during the first fifteen months of the work package (from M6 to M20). 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 implemented 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	04/07/2022	Preliminary Template	RISE
V0.2	30/11/2022	Description of the components	RISE, UWS, MAR, IPN, TRU, XLAB, BOX2M, ATOS
V0.3	05/12/2022	Quality assessment	XLAB
V0.4	16/12/2022	Internal review	UWS
V0.5	23/12/2022	Final editing	XLAB
V1.0	04/01/2023	Project coordinator assessment and final edits	IPN

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

Deliverable 3.2 (Horizontal Planes – second version) is a technical report presenting the intermediate state of the research activities performed around the components in the ARCADIAN-IoT framework, that belong to the horizontal plans (Privacy, Security, and Common planes) and are being developed in Work Package 3 (WP3) of the ARCADIAN-IoT project. The deliverable is an update of Deliverable 3.1, submitted in M20, that presented the preliminary outcomes. Nevertheless, D3.2 is self-contained and provides all the necessary information about the development of the components. Every component is presented in a unified format: the presentation starts with an overview and a recall of respective objectives and target Key Performance Indicators (KPIs), followed by a description of the technology research, where research findings and produced outcomes are described. Additionally, design specifications are given, helping to understand the (internal) technical details of every component, as well as the interfaces to other components. Finally, the results of evaluations and the planned future work 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 focusing 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 identity management 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 **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 **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.

ARCADIAN-IoT

TABLE OF CONTENTS

EXECUTIVE SUMMARY **5**
TABLE OF CONTENTS **6**
LIST OF FIGURES **11**
LIST OF TABLES **13**
ABBREVIATIONS **14**
1 INTRODUCTION **17**
1.1 Objectives 18
2 PRIVACY PLANE **21**
2.1 Self-aware Data Privacy (MAR) 21
2.1.1 Overview 21
2.1.1.1 Description 21
2.1.1.2 Requirements 21
2.1.1.3 Objectives and KPIs 21
2.1.2 Technology research 22
2.1.2.1 Background 22
2.1.2.2 Research findings and achievements 22
2.1.2.3 Produced resources 22
2.1.3 Design specification 23
2.1.3.1 Logical architecture view 24
2.1.3.2 Interface description 26
2.1.3.3 Technical solution 26
2.1.4 Evaluation and results 29
2.1.5 Future work 29
2.2 Federated AI (RISE) 30
2.2.1 Overview 30
2.2.1.1 Description 30
2.2.1.2 Requirements 30
2.2.1.3 Objectives and KPIs 30
2.2.2 Technology research 31
2.2.2.1 Background 31
2.2.2.2 Research findings and achievements 32
2.2.2.3 Produced resources 35
2.2.3 Design specification 36
2.2.3.1 Logical architecture view 36
2.2.3.2 Sequence diagrams 37
2.2.3.3 Interface description 37

2.2.3.4 Technical solution 38
2.2.4 Evaluation and results 38
2.2.5 Future work....38
3 SECURITY PLANE 39
3.1 Network Flow Monitoring (UWS)....39
3.1.1 Overview 39
3.1.1.1 Description 39
3.1.1.2 Requirements 39
3.1.1.3 Objectives and KPIs 39
3.1.2 Technology research 40
3.1.2.1 Background 40
3.1.2.2 Research findings and achievements 40
3.1.2.3 Produced resources....41
3.1.3 Design specification....42
3.1.3.1 Logical architecture view....42
3.1.3.2 Sequence diagrams....43
3.1.3.3 Interface description 43
3.1.3.4 Technical solution....45
3.1.4 Evaluation and results 47
3.1.5 Future work....47
3.2 Behaviour Monitoring (IPN)....48
3.2.1 Overview 48
3.2.1.1 Description 48
3.2.1.2 Requirements 48
3.2.1.3 Objectives and KPIs 48
3.2.2 Technology research 49
3.2.2.1 Background 49
3.2.2.2 Research findings and achievements 51
3.2.2.3 Produced resources....53
3.2.3 Design specification....54
3.2.3.1 Logical architecture view....54
3.2.3.2 Sequence diagrams....55
3.2.3.3 Interface description 56
3.2.3.4 Technical solution....57
3.2.4 Evaluation and results 59
3.2.5 Future work....60
3.3 Cyber Threat Intelligence (RISE) 61
3.3.1 Overview 61

3.3.1.1 Description 61
3.3.1.2 Requirements 61
3.3.1.3 Objectives and KPIs 61
3.3.2 Technology research 62
3.3.2.1 Background 62
3.3.2.2 Research findings and achievements 64
3.3.2.3 Produced resources....65
3.3.3 Design specification....66
3.3.3.1 Logical architecture view....66
3.3.3.2 Sequence diagrams....66
3.3.3.3 Interface description 67
3.3.3.4 Technical solution....68
3.3.4 Evaluation and results 68
3.3.5 Future work....68
3.4 Network Self-protection (UWS) 69
3.4.1 Overview 69
3.4.1.1 Description 69
3.4.1.2 Requirements 69
3.4.1.3 Objectives and KPIs 69
3.4.2 Technology research 70
3.4.2.1 Background 70
3.4.2.2 Research findings and achievements 71
3.4.2.3 Produced resources....75
3.4.3 Design specification....75
3.4.3.1 Logical architecture view....75
3.4.3.2 Sequence diagram....76
3.4.3.3 Interface description 77
3.4.3.4 Technical solution....77
3.4.4 Evaluation and results 78
3.4.5 Future work....79
3.5 IoT Device Self-protection (IPN) 80
3.5.1 Overview 80
3.5.1.1 Description 80
3.5.1.2 Requirements 80
3.5.1.3 Objectives and KPIs 80
3.5.2 Technology research 81
3.5.2.1 Background 81
3.5.2.2 Research findings and achievements 81

3.5.2.3 Produced resources....83
3.5.3 Design specification....83
3.5.3.1 Logical architecture view....83
3.5.3.2 Sequence diagrams....84
3.5.3.3 Interface description....85
3.5.4 Evaluation and results....86
3.5.5 Future work....87
3.6 Network Self-healing (UWS)....88
3.6.1 Overview....88
3.6.1.1 Description....88
3.6.1.2 Requirements....88
3.6.1.3 Objectives and KPIs....88
3.6.2 Technology research....89
3.6.2.1 Background....89
3.6.2.2 Research findings and achievements....90
3.6.2.3 Produced resources....91
3.6.3 Design specification....91
3.6.3.1 Logical architecture view....91
3.6.3.2 Sequence diagrams....91
3.6.3.3 Interface description....92
3.6.3.4 Technical solution....93
3.6.4 Evaluation and results....94
3.6.5 Future work....95
4 COMMON PLANE....96
4.1 Hardened Encryption (XLAB, BOX2M)....96
4.1.1 Overview....96
4.1.1.1 Description....96
4.1.1.2 Requirements....97
4.1.1.3 Objectives and KPIs....97
4.1.2 Technology research....100
4.1.2.1 Background....100
4.1.2.2 Research findings and achievements....101
4.1.2.3 Produced resources....109
4.1.3 Design specification....111
4.1.3.1 Logical architecture view....111
4.1.3.2 Sub-use cases....112
4.1.3.3 Sequence diagrams....113
4.1.3.4 Interface description for Hardened encryption with RoT signatures....115

4.1.3.5 Technical solution – Hardened Encryption with RoT signatures 116
4.1.3.6 Technical solution – Hardened Encryption with cryptochip system 118
4.1.4 Evaluation and results 120
4.1.5 Future work 122
4.2 Permissioned Blockchain (ATOS) 124
4.2.1 Overview 124
4.2.1.1 Description 124
4.2.1.2 Requirements 124
4.2.1.3 Objectives and KPIs 124
4.2.2 Technology research 125
4.2.2.1 Background 125
4.2.2.2 Research findings and achievements 125
4.2.2.3 Produced resources 129
4.2.3 Design specification 129
4.2.3.1 Sub-use cases 129
4.2.3.2 Logical architecture view 131
4.2.3.3 Interface description 132
4.2.3.4 Technical solution 133
4.2.4 Evaluation and results 137
4.2.5 Future work 138
5 CONCLUSIONS 139
REFERENCES 140

LIST OF FIGURES

Figure 1 - ARCADIAN-IoT Conceptual representation highlighting Horizontal Planes 17
Figure 2 - SADP1 Preliminary logical architecture for the Self-Aware Data Privacy component 23
Figure 3 - SADP2 Self-Aware Data Privacy logical architecture exploded 24
Figure 4 - SADP3 Anubis logical architecture 24
Figure 5 - SADP4 Amon logical architecture... 25
Figure 6 - SADP5 Logical view of the combined action of Anubis and Amon 26
Figure 7 - SADP6 Policy example 27
Figure 8 - SADP 7 Example of GUI for policy definition 29
Figure 9 - Overview of Data rebalancing approach 33
Figure 10 - The overview of subcomponent (Model resizing and sharing) 35
Figure 11 - Component architecture of Federated AI 36
Figure 12 - Sequence diagram 37
Figure 13 - 5G multi-stakeholder network segments... 40
Figure 14 - Snort IDS event against all overlay network information from the flow 41
Figure 15 - Logical architecture of the Network Flow Monitoring... 42
Figure 16 - Sequence diagram of the Network Flow Monitoring component 43
Figure 17 - Deployment architecture view of the Network Flow Monitoring component............. 46
Figure 18 - Detailed architecture of the system call-based HIDS solution 52
Figure 19 - Example of a raw system call log extracted with perf.. 54
Figure 20 - Behaviour Monitoring component architecture.. 55
Figure 21 - Device Behaviour Monitoring Sequence Diagram 56
Figure 22 - On device deployment architecture view 57
Figure 23 - Non-device deployment architecture view 58
Figure 24 - The architecture of CTI.. 66
Figure 25 - Sequence diagram 67
Figure 26 - Overview of the 5G IoT integrated in the Network Self-Protection 73
Figure 27 - Network Self-Protection architectural overview.. 75
Figure 28 - Network Self-Protection architectural overview.. 76
Figure 29 - Network Self-Protection architectural overview.. 78
Figure 30 - MAPE-K Architecture Representation.. 82
Figure 31 - Policy structure example of Device Self Protection.. 83
Figure 32 - Device Self-Protection component architecture 84
Figure 33 - Expected behaviour of the IoT Device Self Protection 85
Figure 34 - Architecture design of the UWS Resource Inventory Agent..................................... 90
Figure 35 - Architecture of the Network Self-healing component. 91

Figure 36 - Sequence diagram of the Network Self-healing component 92
Figure 37 - Deployment architecture view of the Network Self-healing component.................. 94
Figure 38 - Research hypothesis for the Hardened Encryption to join ABE with eSIM as RoT 102
Figure 39 - The encryption and decryption process in HE 103
Figure 40 eUICC structure comprising the IoT SAFE [16] 104
Figure 41 - ARCADIAN-IoT Hardened Encryption prototype mapped into GSMA IoT SAFE architecture 105
Figure 42 - Multi-authority key management.. 107
Figure 43 - Hardware high level architecture 108
Figure 44 - Middleware architecture 109
Figure 45 - Hardware prototypes... 111
Figure 46 - Logical architecture view of HE 111
Figure 47 - Crypto chip system architecture 112
Figure 48 - The sequence diagram of the HE component with internal and external interfaces 114
Figure 49 - Device onboarding 115
Figure 50 - Device traffic management.. 115
Figure 51 - Grid domain implementation of crypto chip system... 116
Figure 52 - Snippets from the HE documentation 118
Figure 53 - Crypto chip system – keys set-up (synced with device firmware provisioning)...... 119
Figure 54 - Crypto chip system - middleware API set-up 119
Figure 55 - IoT stress simulator 122
Figure 56 - Example of a blockchain network with three peer nodes. 126
Figure 57 - Publish ARCADIAN-IoT Data to Permissioned Blockchain 130
Figure 58 - Read published ARCADIAN-IoT Data Objects 130
Figure 59 - Write and Fetch Sidetree blockchain transactions 131
Figure 60 - Logical architecture for publishing ARCADIAN-IoT Objects to Permissioned Blockchain.. 132
Figure 61 - Publish ARCADIAN-IoT Data Objects Interface Description 133
Figure 62 - Permissioned Blockchain for Publishing ARCADIAN-IoT Data deployment view.. 134
Figure 63 - Reputation Private Data Collection 136
Figure 64 - ABE Key Private Data Collection... 136

LIST OF TABLES

Table 1 - Mapping the components to WP tasks 18
Table 2 - Progres on the external interfaces of the Federated AI component 26
Table 3 - Progres on the external interfaces of the Federated AI component 38
Table 4 - Comparison between SIDS and AIDS 50
Table 5 - Behaviour Monitoring External Interfaces 57
Table 6 - Device Behaviour Monitoring Output Specification 58
Table 7 - Model Training Results 59
Table 8 - FL Model Training Results 60
Table 9 - The external interfaces of the CTI component 68
Table 10 - Proposed new Open Flow fields 74
Table 11 - Parsing of encapsulation, tunneling and tagging protocols: tunnel type values and . 74
Table 12 - Device Self-Protection External Interfaces 86
Table 13 - Device Self-Protection Output Specification 86
Table 14 - Methods from the first prototype of ARCADIAN-IoT security eSIM applet made accessible by the existent device middleware 106
Table 15 - Encryption and decryption overhead for using ABE 120
Table 16 - ARCADIAN-IoT Smart Contract GW Interface Authorisation Permissions 135

ABBREVIATIONS

3PP	3rd Party Provider
4G	4th Generation
5G	5th Generation
5GS	5G System
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
eSIM	embedded SIM
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
ID	Identifier
IDS	Intrusion Detection System
IID	Independent and Identically Distributed
IMSI	International Mobile Subscriber Identity
IoC	Indicator of Compromise
IoT	Internet of Things
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
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
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
VISP	Virtualisation Infrastructure Service Provider
VLAN	Virtual Local Area Network
VxLAN	Virtual Extensible Local Area Network
WP	Work Package
XDP	eXpress Data Path
XML	eXtensible Markup Language

1 INTRODUCTION

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 together 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).

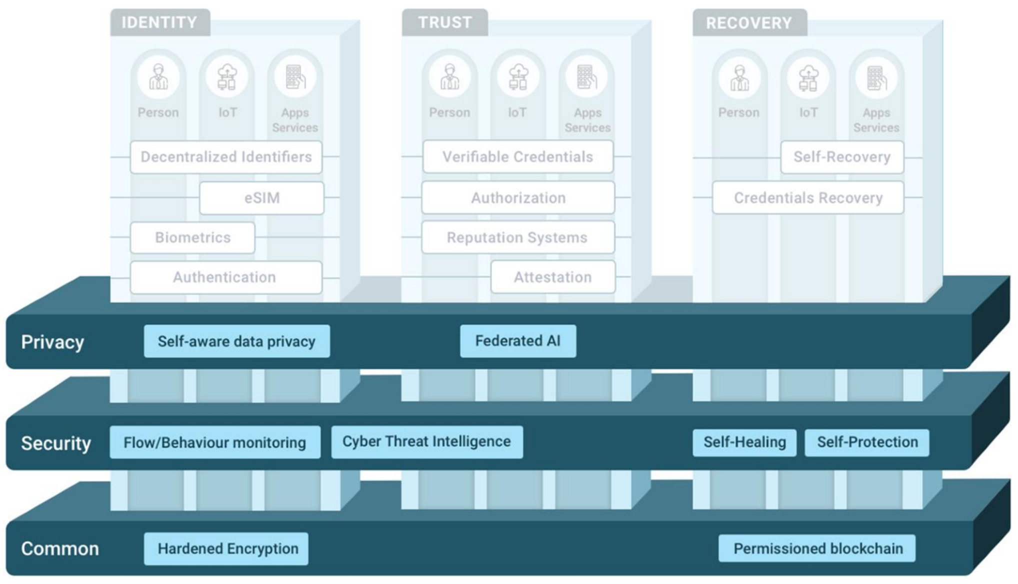


Figure 1 - ARCADIAN-IoT Conceptual representation highlighting Horizontal Planes

Work Package 3 (WP3) in the ARCADIAN-IoT project is dedicated to the design and technological development of the functionalities that are mapped into the Horizontal Planes for each selected use case. It is organized in five tasks, each one focusing on one or more components. The research activity in WP3 is being conducted from October 2021 to October 2023 and this deliverable (D3.2) details the research activities, the provided resources, and the results of evaluations, that have been obtained within WP3 until December 2022 (fifteen months). The reporting takes an agile approach: the information in D3.2 is an update of the status reported in D3.1 and is self-contained. In addition to providing the current status in the research of the beforementioned technologies, this deliverable focuses also on providing technical information, such as design specifications, and interface description, to help the integration among the components. The document is supported with implemented source code (by links in the text) that has been produced during the reporting period if the sharing is aligned with the partners' exploitation strategy and the code is not subjected to any Intellectual Property Rights (IPR) restrictions. Furthermore, since the development of the use-cases in WP5 is already in progress, the deliverable will boost the integration and validation process. The final WP3 deliverable D3.3 will provide the final description of the components and the final results of their evaluation.

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 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.

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	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. Information on the planned future work.

1.1 Objectives

WP3 aims at contributing to the achievements of six main objectives in the ARCADIAN-IoT projects, defined in the ARCADIAN-IoT grant agreement. Each objective comes with individual Key Performance Indicators (KPIs), that have been 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. We list the objectives and their KPIs here, while the further details on the target and current values are given in the descriptions 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 techniques 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.**
  - 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.
  - Add Root of Trust information to encrypted data with eSIM based 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.
    - 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.

2 PRIVACY PLANE

2.1 Self-aware Data Privacy (MAR)

2.1.1 Overview

2.1.1.1 Description

Within the scope of the ARCADIAN-IoT project, Martel is developing 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 relates 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

KPI scope	
Number of policies handled by the protection algorithm	
Measurable Indicator	
Performance of the algorithms to secure the data (e.g., speed or robustness).	
Target value (M30)	***Current value (M20)***
5 encryption/anonymisation	1 anonymisation policy implemented

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 (M30)	***Current value (M20)***
≥80%	n/a as the component is still under

	development

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 (M30)	***Current value (M20)***
90% Positive Feedback	n/a

2.1.2 Technology research

2.1.2.1 Background

As illustrated in D3.1, the investigation had an applicative focus on state-of-the-art algorithms for the efficient implementation of policy management and enforcement.

2.1.2.2 Research findings and achievements

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

Open Policy Agent (OPA[1]), 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[2], 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.

These two tools are combined and concatenated in two sub-modules: Anubis and Amon which together constitute the policy management module of the Self-aware data privacy.

Anubis[3] 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[4], which is instead the project’s foreground and allows to: 1) define anonymization and encryption policies for data and 2) apply encryption (and decryption policies) on data.

2.1.2.3 Produced resources

In Figure 2 *SADP1* we recall the logical architecture 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.

Bullet 1 to 3 are completed, while we are currently working in integrating advanced technique of encryption into the Amon sub-module (4), leaving for the second stage the recommender system and GUI implementation.

(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.

[1] https://www.openpolicyagent.org
[2] https://www.envoyproxy.io/docs/envoy/latest
[3] https://github.com/orchestracities/anubis
[4] https://github.com/orchestracities/amon

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 will be implemented in the second stage. The GUI will leverage and extend the one already defined for Anubis and illustrated below.

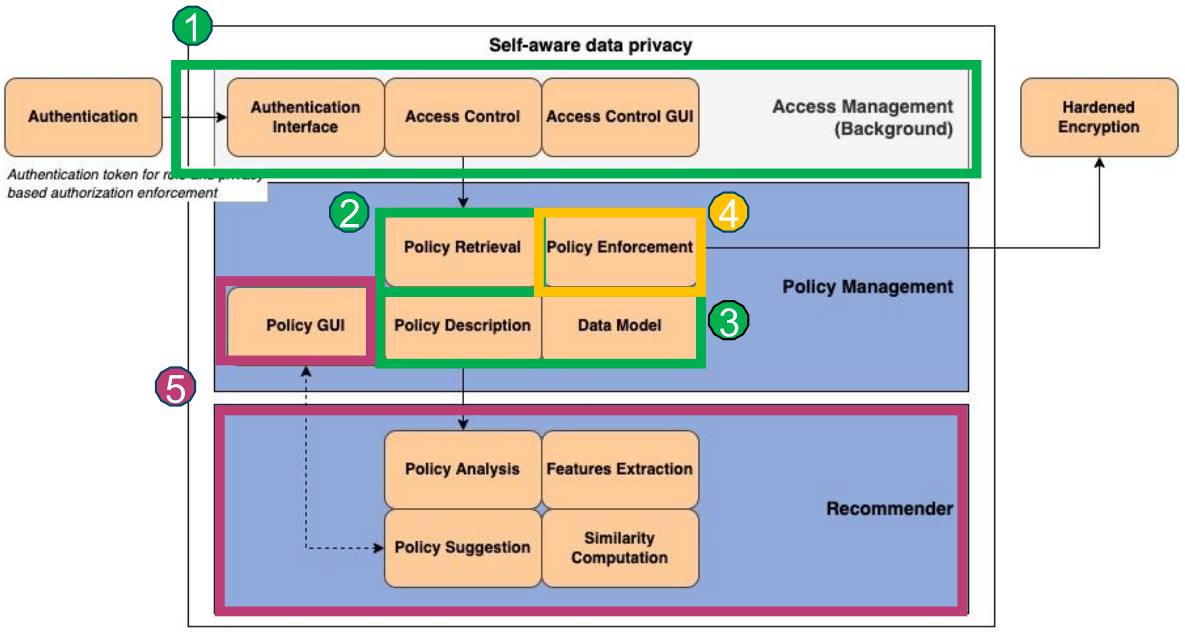


Figure 2 - Preliminary logical architecture for the Self-Aware Data Privacy component

2.1.3 Design specification

In Figure 3 *SADP2* we illustrated the two sub-modules Anubis and Amon in relation with the logical architecture presented in D3.1 and provide a better inside on the interaction of the different components. This is described in the next sections into more details.

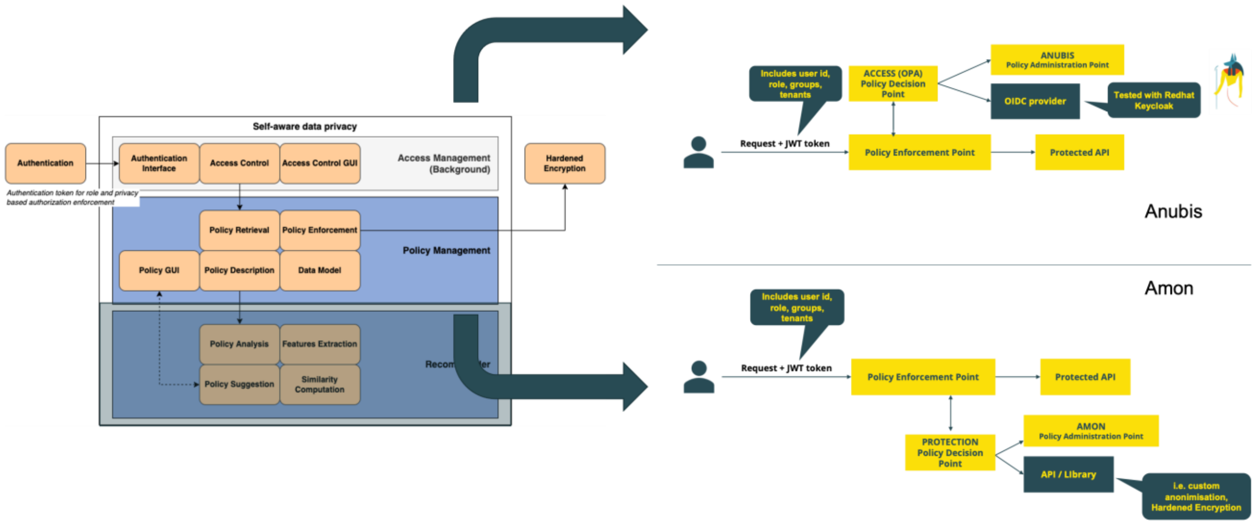


Figure 3 - Self-Aware Data Privacy logical architecture exploded

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. The figure below shows the current architecture (Figure *SADP3*).

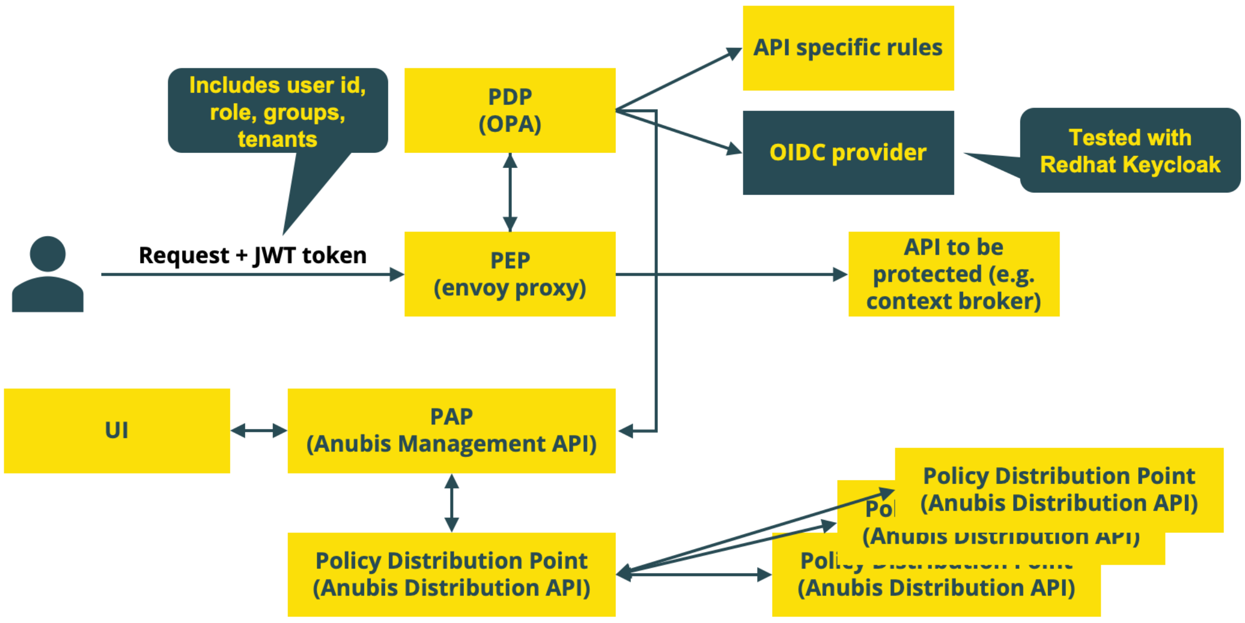

Includes user id, role, groups, tenants
PDP (OPA)
API specific rules
OIDC provider
Tested with Redhat Keycloak
Request + JWT token
PEP (envoy proxy)
API to be protected (e.g. context broker)
UI
PAP (Anubis Management API)
Policy Distribution Point (Anubis Distribution API)
Policy Distribution Point (Anubis 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 WAC 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[5], Anubis management, and JWT-based authentication. In addition to that, thanks to the libp2p middleware[6], polices can also be distribute across differed 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. The figure below (SADP4) shows Amon's logic architecture.

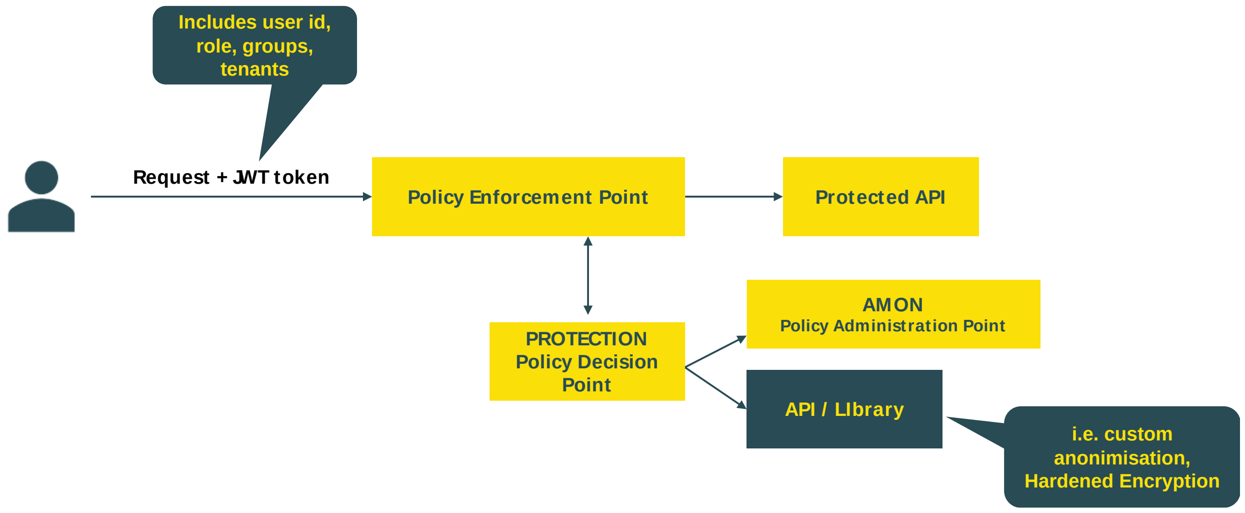

Includes user id, role, groups, tenants
Request + JWT token
Policy Enforcement Point
Protected API
AMON
Policy Administration Point
PROTECTION
Policy Decision Point
API / LIbrary
i.e. custom anonimisation, Hardened Encryption

Figure 5 - Amon logical architecture

In the figure below we illustrate the joint action Anubis-Amon, in particular by the Policy Enforcement Point which is able to concatenate the chain of authorization first for access (Anubis), then for data protection (Amon).

[5] https://fiware-orion.readthedocs.io/en/master/
[6] https://libp2p.io/

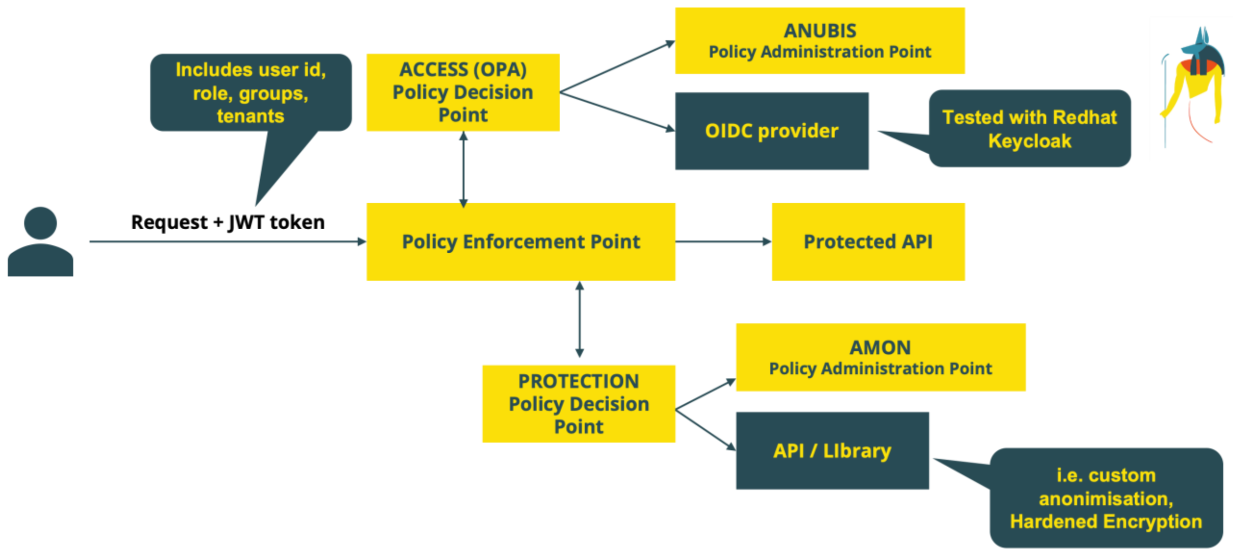

ANUBIS
Policy Administration Point
Includes user id, role, groups, tenants
ACCESS (OPA) Policy Decision Point
OIDC provider
Tested with Redhat Keycloak
Request + JWT token
Policy Enforcement Point
Protected API
AMON
Policy Administration Point
PROTECTION Policy Decision Point
API / LIbrary
i.e. custom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

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

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

The PEP pass over the request to the PDP (Policy Decision Point), 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 (Policy Administration Point), 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 progress on the external interfaces:

Partner	Definition	Channel	Exchange	Status
XLAB	Hardened Encryption	Direct Code Integration	Encryption libraries integrated inside Amon	In progress
IPN	Authentication	Direct Code Integration	JWT Token respecting the OIDC standard	Pending

Table 2 - Progres on the external interfaces of the SADP component

The external interfaces that have the status of ‘Pending’ refer to component integration that is relative to the second prototype, and thus, the efforts for integration of the two components have not yet started.

2.1.3.3 Technical solution

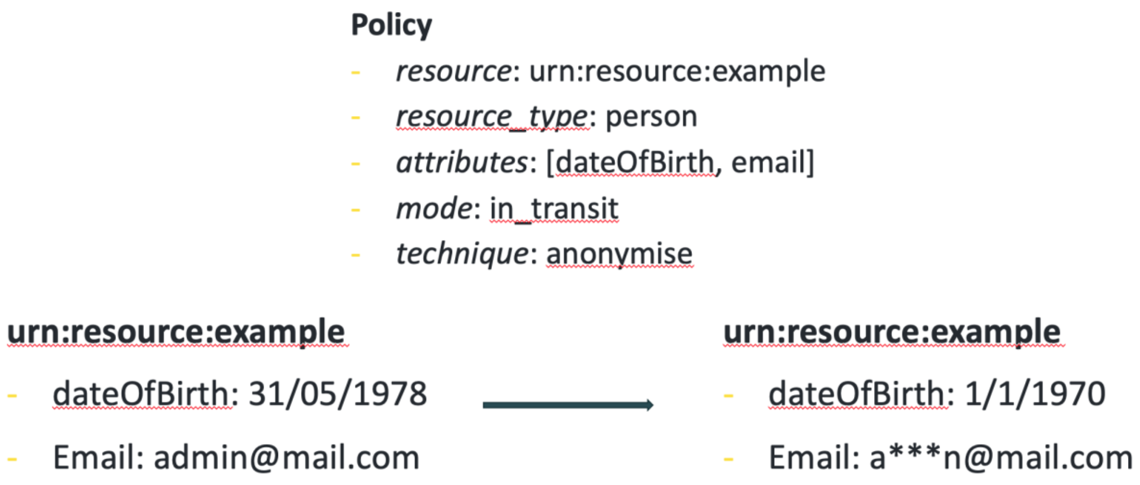
We present in the next sections the properties of the policies with an example and summarise how the API are structured.

2.1.3.3.1 Policy Format

The formal data protection policy specification is defined by an oc-acl vocabulary[7]. 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 the figure below 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[8].

The Amon APIs depends 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[9].

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

[7] https://github.com/orchestracities/anubis-vocabulary/blob/master/oc-acl.ttl
[8] https://github.com/orchestracities/amon/tree/master/amon-management-api
[9] https://fiware-orion.readthedocs.io/en/3.7.0/user/multitenancy.html

related discussion in FIWARE Orion Context Broker documentation[10].

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

  <u>note</u>: 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
  ```

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

Generate UML diagram for data models

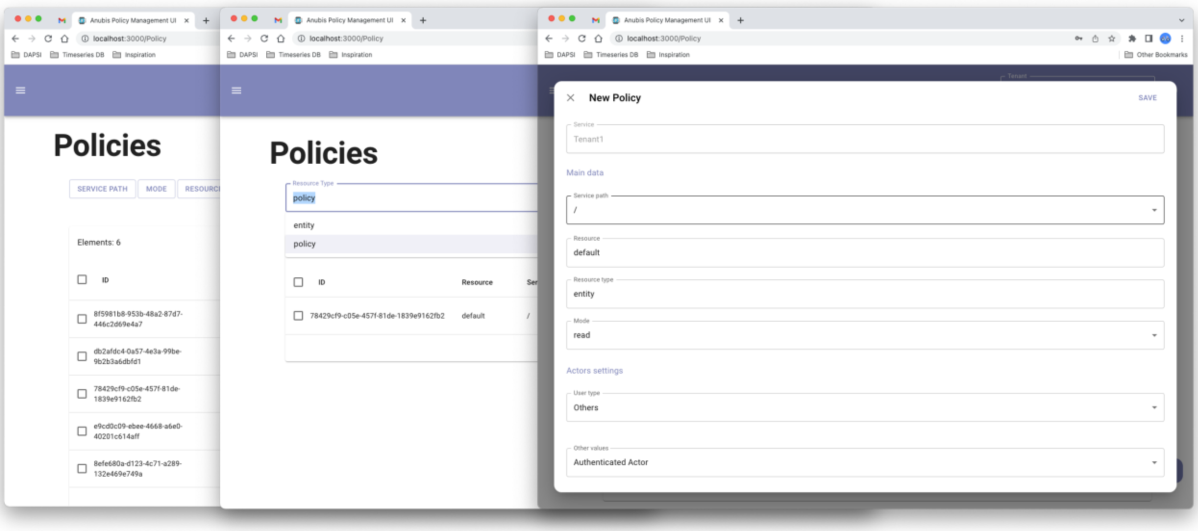
```
$ python generate_uml.py
```

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 will be developed similarly to the one developed for Anubis and reported below as an example.



Elements: 6

ID
8f5981b8-953b-48a2-87d7-446c2d69e4a7
db2afdc4-0a57-4e3a-99be-9b2b3a6dbfd1
78429cf9-c05e-457f-81de-1839e9162fb2
e9cd0c09-ebee-4668-a6e0-40201c614aff
8efe680a-d123-4c71-a289-132e469e749a

Resource Type: policy (entity, policy)

ID	Resource	Ser
78429cf9-c05e-457f-81de-1839e9162fb2	default	/

New Policy

Service: Tenant1

Main data

Service path: /

Resource: default

Resource type: entity

Mode: read

Actors settings

User type: Others

Other values: Authenticated Actor

Figure 8 - Example of GUI for policy definition

2.1.4 Evaluation and results

The integration is in progress, and we are currently evaluation the technical functionality of the modules.

2.1.5 Future work

As highlighted at the beginning of the chapter, the next steps will be the consolidation of the integration between Anubis and Amon, the validation of the Hardened Encryption as one of the potential algorithms for data protection and the definition and implementation of the recommender system, which is the last component of the Self-Aware Data Privacy pending and due to M36.

2.2 Federated AI (RISE)

2.2.1 Overview

2.2.1.1 Description

Within the scope of the ARCADIAN-IoT project, Research Institutes of Sweden (RISE) is building dependable and privacy preserving Federated Learning (FL) capabilities which will be deployed in machine learning (ML)-based components (e.g., CTI and Behaviour Monitoring components). The proposed solution will provide 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 is investigating 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 is being 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 will implement 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 will be provided by (i) the data rebalancer which will only share generated synthetic data and (ii) the model resizing and sharing subcomponent which will use 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 will be 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 (M30)	***Current value (M20)***
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 (M30)	***Current value (M20)***
AUC > 99%, F1-score > 94% #epochs to converge < 35 epochs	n/a as the subcomponent is still under development

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 (M30)	***Current value (M20)***
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%	n/a as the component is still under development

2.2.2 Technology research

2.2.2.1 Background

During the current reporting period, RISE has completed the development of the data rebalancer, which aims at rebalancing non-IID and imbalanced data in a federated set-up. RISE has also worked on the design and implementation of model resizing and sharing subcomponent. The research activity performed so far 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 a novel communication-efficient and robust aggregation rule for FL.

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

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 9 depicts how the proposed Data rebalancer subcomponent works.

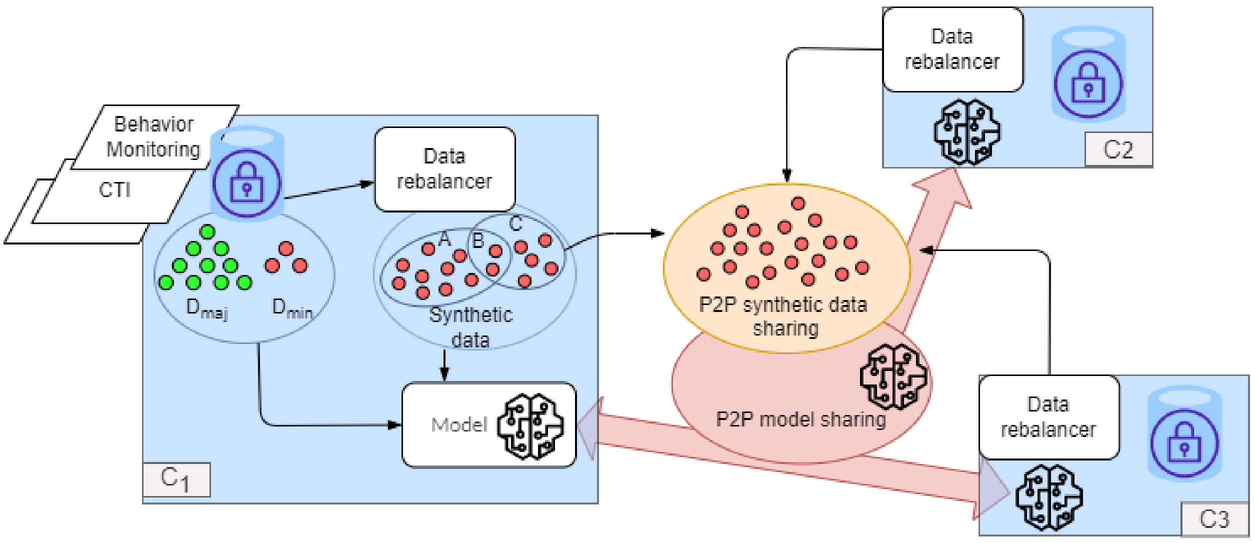


Figure 9 - Overview of Data rebalancing approach

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 $(A \cup B)$[11] 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.

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

```
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 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.

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 10 depicts how the proposed model resizing and sharing subcomponent works.

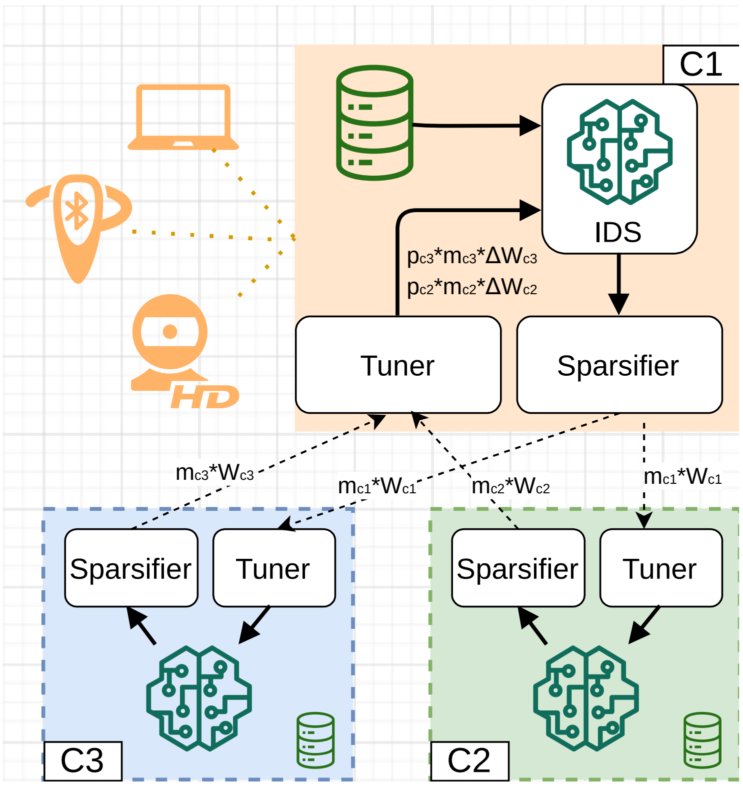


Figure 10 - The overview of subcomponent (Model resizing and sharing)

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 10, 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 decide 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.

2.2.2.3 Produced resources

The developed code for data rebalancer 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. 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. Update the local model: The model is updated by FedAvg or Weighted FedAvg

Structure of the repository:

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

2.2.3 Design specification

2.2.3.1 Logical architecture view

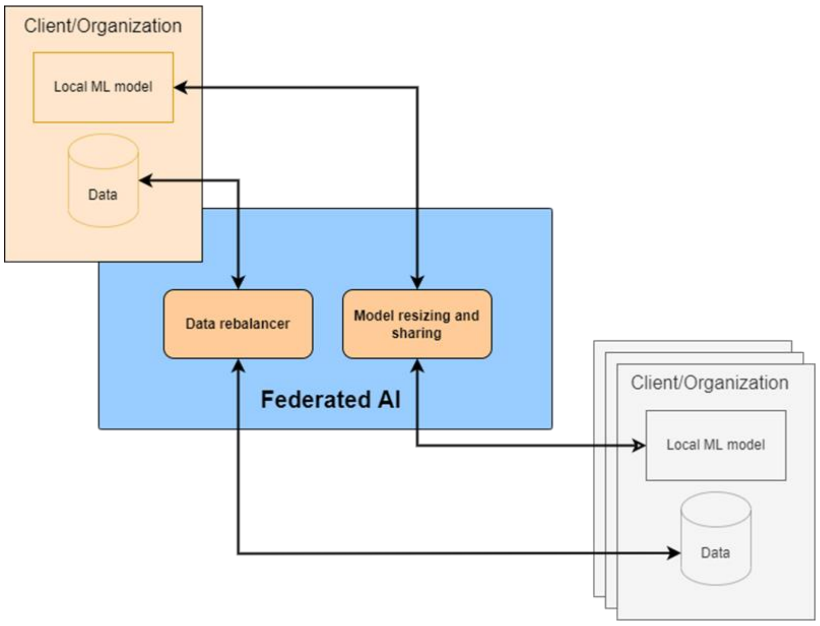


Figure 11 - Component architecture of Federated AI

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 11. 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 12 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.

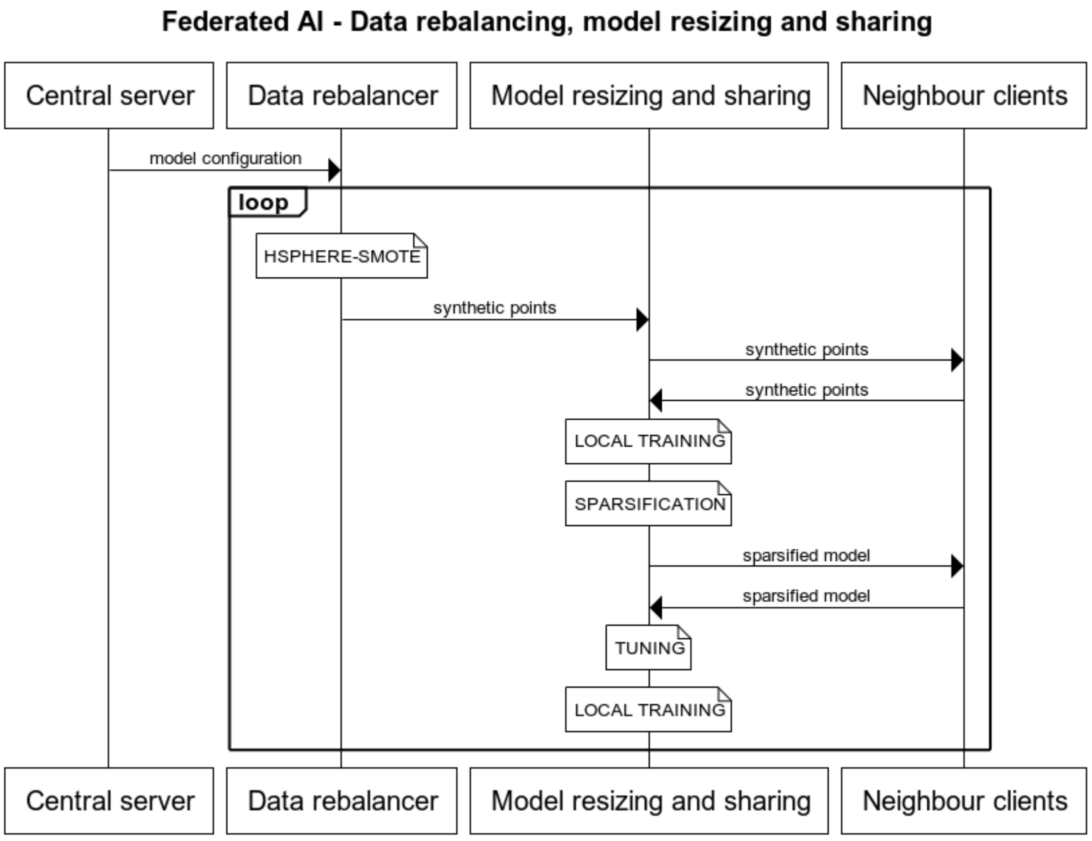


Figure 12 - Sequence diagram

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 details the progress on the external interfaces:

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

Table 3 - Progres on the external interfaces of the Federated AI component

The external interfaces that have the status of 'Pending' refer to component integration that is relative to the second prototype, and thus, the efforts for integration of the two components have not yet started.

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-sourced 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 is evaluating the robustness of HSphere-SMOTE against privacy breach attacks (the detailed results will be presented in a research paper that is in progress).

2.2.5 Future work

Future work will focus on the following: (i) Complete the Development of the method for model resizing and sharing and performance evaluation, (ii) Evaluate the robustness of HSphere-SMOTE against privacy breach attacks, (iii) Integration and deployment of the FL component and its subcomponents into the Behaviour Monitoring and CTI components. The final prototype will be available at the end of the development cycle and provided in deliverable D3.3.

3 SECURITY PLANE

3.1 Network Flow Monitoring (UWS)

3.1.1 Overview

3.1.1.1 Description

Within the scope of the ARCADIAN-IoT project, the University of West of Scotland (UWS) is developing the Network Flow Monitoring (NFM) component that will act 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 will be 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 will act as a wrapper for the NIDS alert and will provide fine data information about the malicious flow detected. This component will provide 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.	
Target value (M30)	***Current value (M20)***
>= 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.	
Target value (M30)	***Current value (M20)***
>= 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) [viii]. 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 to create 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 13 - 5G multi-stakeholder network segments.

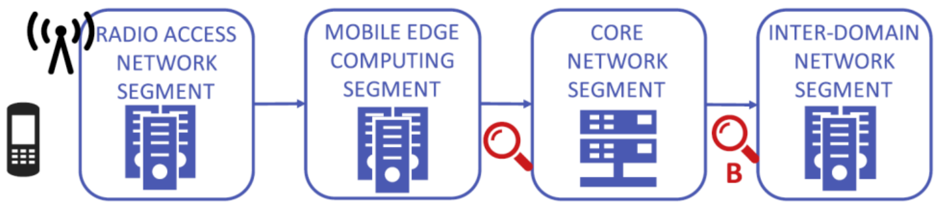
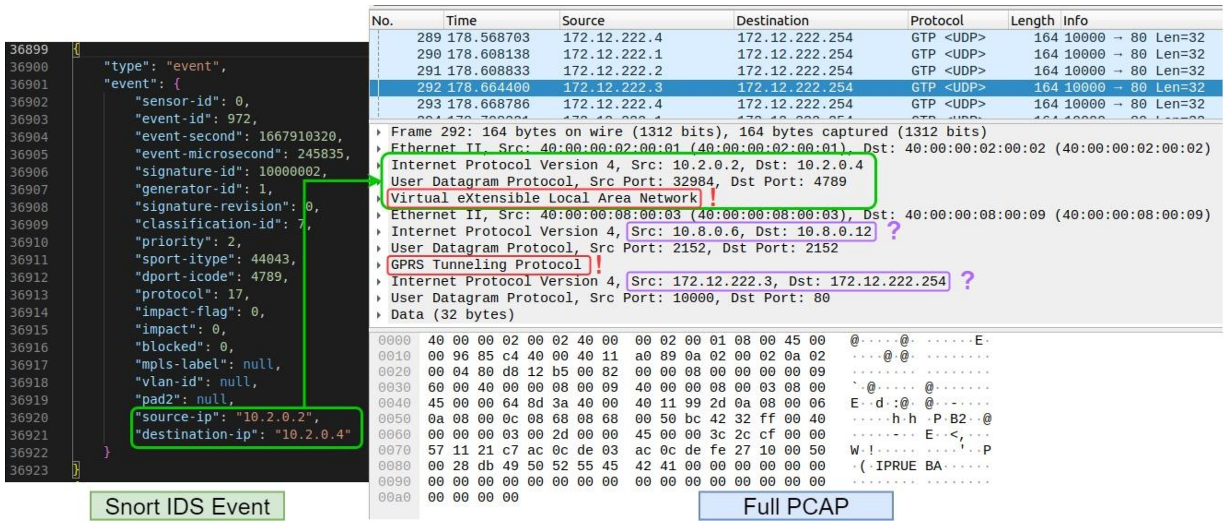

RADIO ACCESS NETWORK SEGMENT
MOBILE EDGE COMPUTING SEGMENT
CORE NETWORK SEGMENT
B
INTER-DOMAIN NETWORK SEGMENT

Figure 13 - 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 are 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 14shows an example of the information that a Snort IDS Event is able to retrieve (see left in Figure 14). 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 14). 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 14 - 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 with support for several different encapsulation levels such as VXLAN, GTP, GRE and/or GENEVE as an extension of traditional NIDS solutions for all network segments in a 5G-IoT infrastructure.

  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

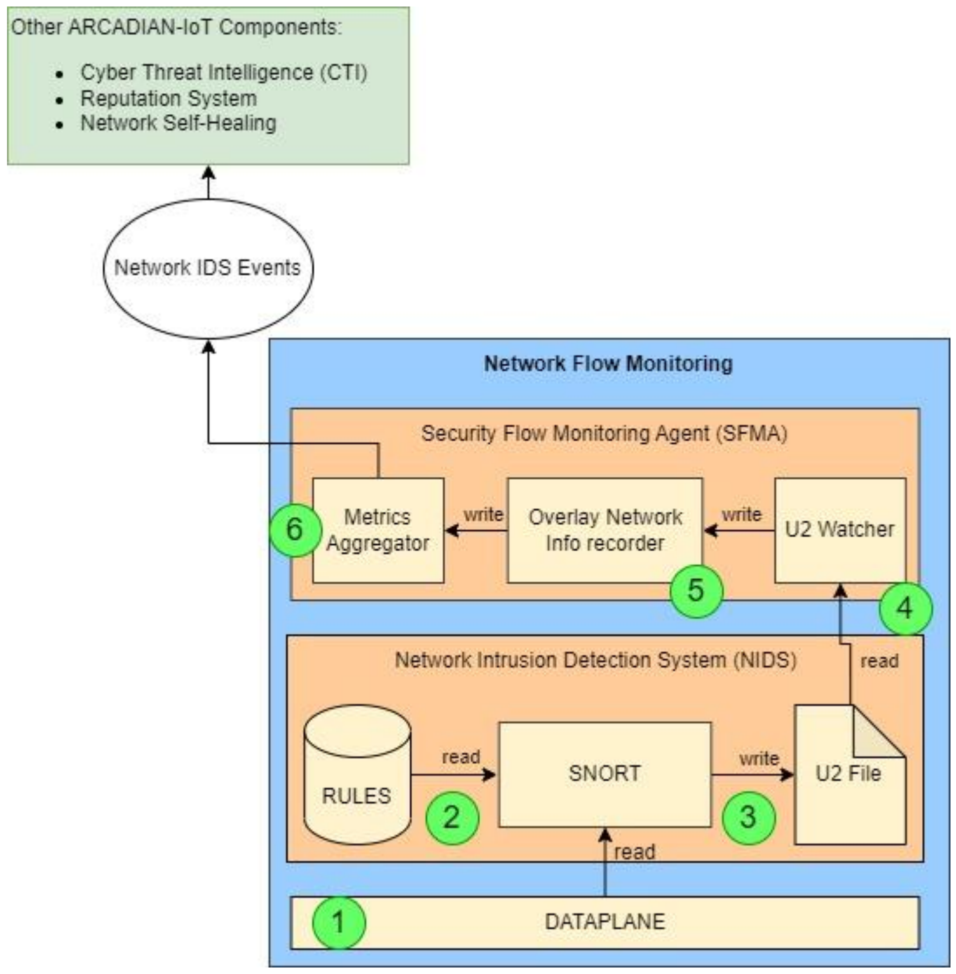

Other ARCADIAN-IoT Components:
Cyber Threat Intelligence (CTI)
Reputation System
Network Self-Healing
Network IDS Events
Network Flow Monitoring
Security Flow Monitoring Agent (SFMA)
Metrics Aggregator
write
Overlay Network Info recorder
write
U2 Watcher
6
5
4
Network Intrusion Detection System (NIDS)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

Figure 15 - Logical architecture of the Network Flow Monitoring

Figure 15 describes the logical architecture that follows the Network Flow Monitoring component. The workflow of the component starts with reading the data plane in the network to which the NFM is attached (see 1 in Figure 15). A Network Intrusion Detection System (NIDS) is reading each packet in that data plane and will be executing a set of security rules (see 2 in Figure 15). When a rule matches with the packet inspected, the NIDS will write a new event in a Unified2 log file with information of the threat (see 3 in Figure 15). 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), which 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 15). 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 15). 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 15).

3.1.3.2 Sequence diagrams

Figure 16 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.

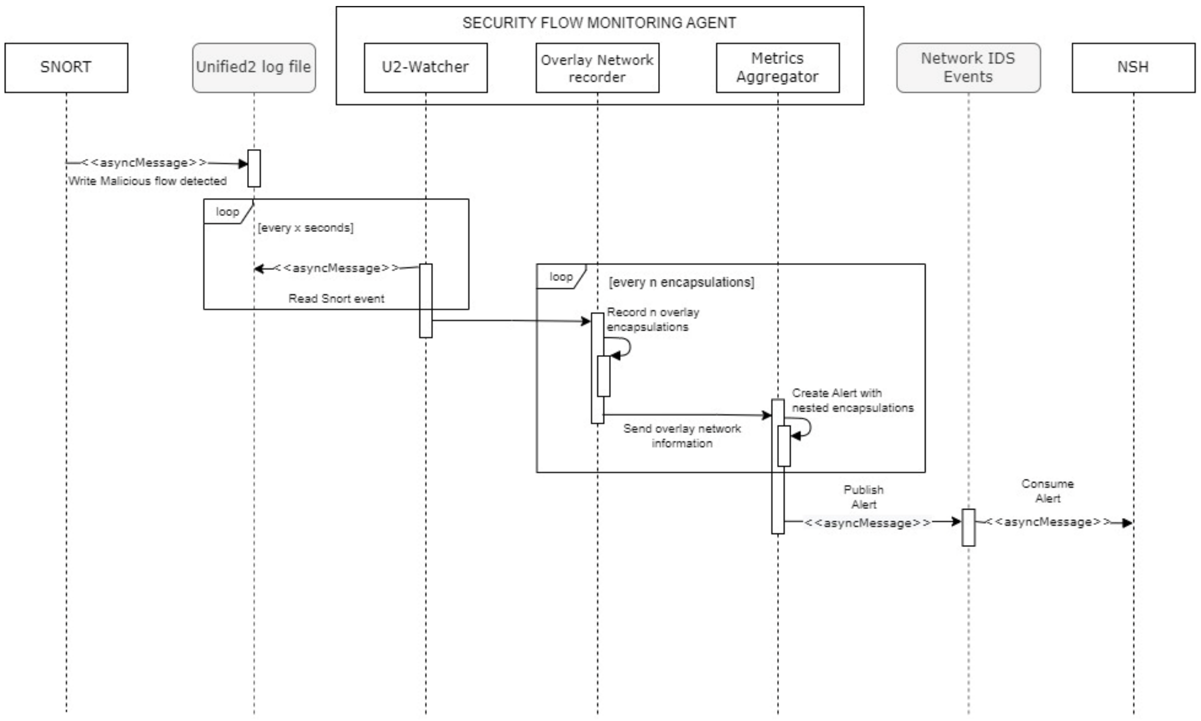


Figure 16 - Sequence diagram of the Network Flow Monitoring component

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	Status
UWS	Network Self-Healing	RabbitMQ AMQP 0.9.1	Network IDS Events	Done
UWS	Network Self-Protection	RabbitMQ AMQP 0.9.1	Network Self-Protection Confirmations	Done
UC	Reputation System	RabbitMQ AMQP 0.9.1	Network IDS Events	In progress

RISE	Cyber Threat Intelligence	RabbitMQ AMQP 0.9.1	Network IDS Events	In progress

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 messagebus. 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 17 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.

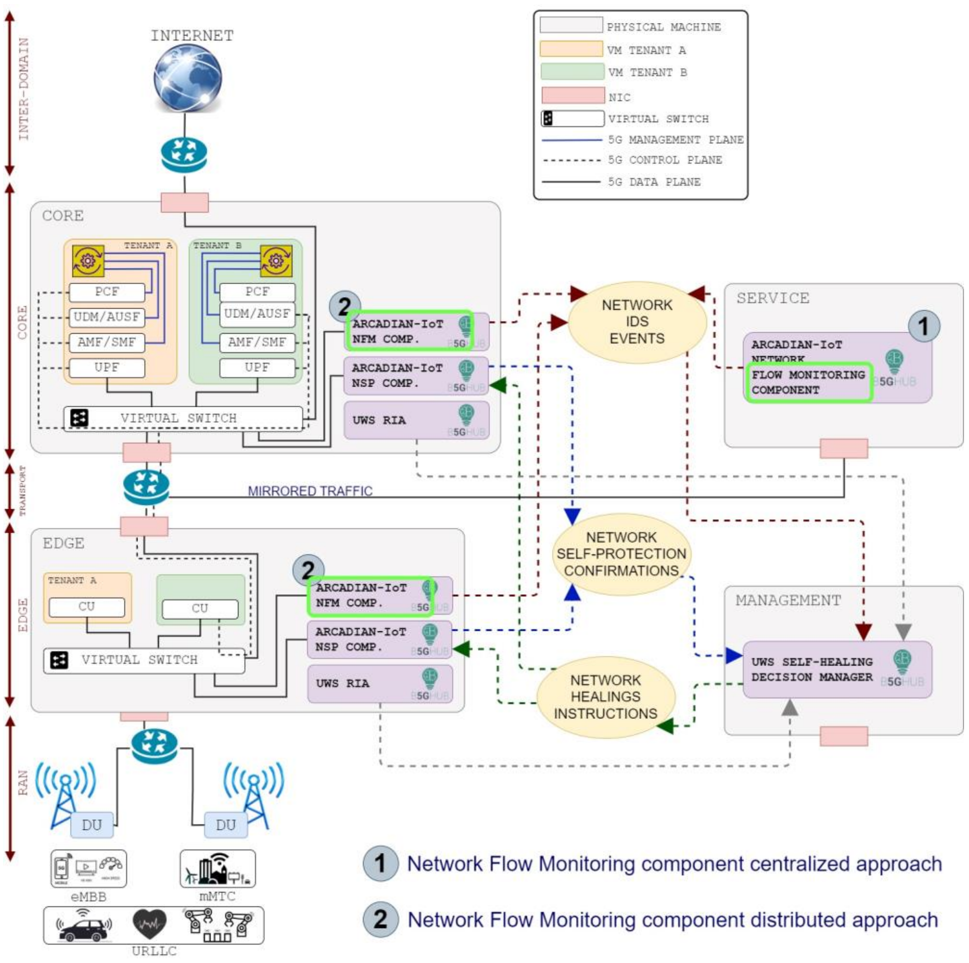


Figure 17 - Deployment architecture view of the Network Flow Monitoring component

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

Following the prototyping and as preliminary step towards the integration with the rest of the architectural components composing the different Horizontal Planes of the ARCADIAN-IoT framework, the overall behaviour of the NFM component will be empirically tested. The tests will be carried out in the cloud data centre at the UWS facilities and will be focused on two main aspects: functional validation and overall performance.

For validating the functionality of the component, it will be exposed to a set of scenarios, in a 5G system, where different known DDoS attacks will be launched. This will ensure that the effectiveness of the component is as expected, as the component is designed to trigger alerts in the moment that one of the network flows matches any of the rules stored into the Snort rules database.

On the other hand, the NFM component will be critically analysed in different areas to demonstrate its performance in different situations. In particular we will analyse:

- **Scalability –** Maximum Number of IoT devices traffic handled.
- **Bandwidth** – Maximum data rate that the component is able to deal with.
- **Delay** – Average time to process each alert; This involves the time consumed by Snort to match a rule and trigger the alert to the unified2 log file, the time consumed by the SFMA to analyse the alert and trigger the new one with the full information of the overlay networks to the following components.

3.1.5 Future work

The UWS team is currently working on the integration between the three components that belong to the cognitive self-protection network security loop. The development of the first version of the NFM component prototype is achieved and is being integrated with the rest of the ARCADIAN-IoT framework's components

After prototyping and as a last step in the development of the component, the prototype will be empirically validated and evaluated (as described in previous Subsection 3.1.4). Hence, an empirical validation and evaluation of the overall performance of this component will be reported in deliverable D3.3 with the subsequent integration in the overall ARCADIAN-IoT framework.

3.2 Behaviour Monitoring (IPN)

3.2.1 Overview

3.2.1.1 Description

To enhance the security of the IoT devices, one of the ARCADIAN-IoT project's objectives is to develop a component that aims to detect anomalous behaviour that occurs on device level. The component at hand is a host-based intrusion detection system (HIDS) specialized for IoT devices, which takes the task of examining incoming events that are specific to the host device (such as what applications are being used, what files are being accessed, permission changes, sequences of system calls and authentication attempts).

This component – Behaviour Monitoring - is comprised by a set of subcomponents that collect and classify the events with the use of lightweight ML models. The models are updated via a FL scheme that ensures privacy preservation of the devices.

The intrusion detection is performed in real time to make sure that the device is operating normally and detect signs of intrusion upon abnormal operational behaviour. The resulting output of this component consists of a value that indicates if the event was classified as an intrusion (i.e., the activity relates to an ongoing intrusion, or intrusion attempt) or normal behaviour, with a confidence level associated with the prediction. The resultant information is then encapsulated and forwarded to other ARCADIAN-IoT components via a message bus (e.g., if there was an intrusion detected, the information should be received by the IoT Device Self-protection component, which has the role of protecting devices against the incoming attacks).

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
Continuous training of IDS models in IoT devices
Measurable Indicator
Performance of the component in IoT devices (i.e., computing resources) Model performance (e.g., TPR)
Benchmarking

Accuracy >= 0.96 TPR >= 0.95 FPR <=0.05 F1-Score >= 0.96 [12]	
Target value (M30)	***Current value (M20)***
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 (M30)	***Current value (M20)***
Consider at least 3 types of input (e.g., syscall sequences, reputation score, authentication results)	Considers at least 1 type of inputs (syscalls)

KPI scope	
Deployment in heterogenous devices	
Measurable Indicator	
Number supported devices	
Target value (M30)	***Current value (M20)***
Support at least 2 types of devices (e.g., drone device, smartphone, industrial IoT device – developed by BOX2M)	Support at least 1 type of device (e.g., drone)

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

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 4 - 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

[12] 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

detect zero-day attacks, contrary to AIDS. On the other hand, anomaly detection requires characterizing and identify 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 4 - 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.

Considering 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.[13] with the following advantages:

- System calls (i.e., root processes) are more advantageous than a typical user process

[13] 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

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.[14] 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 [15]. 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 [16] 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 solution employs a HIDS approach complemented by a Federated Learning design.

3.2.2.2 Research findings and achievements

The approach followed for the development of the behaviour monitoring component started with the validation of Machine Learning (ML) models on an open-source system call dataset known as ADFA-LD[17]. 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.

[14] W. Lee and S. Stolfo. Data mining approaches for intrusion detection. In Proceedings of the 7th USENIX Security Symposium, 1998.
[15] P. Helman and J. Bhangoo. A statistically based system for prioritizing information exploration under uncertainty. Trans. Sys. Man Cyber, 1997.
[16] Yihua Liao and Rao Vemuri. Using text categorization techniques for intrusion detection, 2002.
[17] https://research.unsw.edu.au/projects/adfa-ids-datasets

Our approach to training and testing 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 represent 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 collect and aggregate the data from (currently, emulated) devices, process it and then train the models with said data with both normal and abnormal device behaviour. Having produced the trained model that has a desired accuracy metric, 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 18.

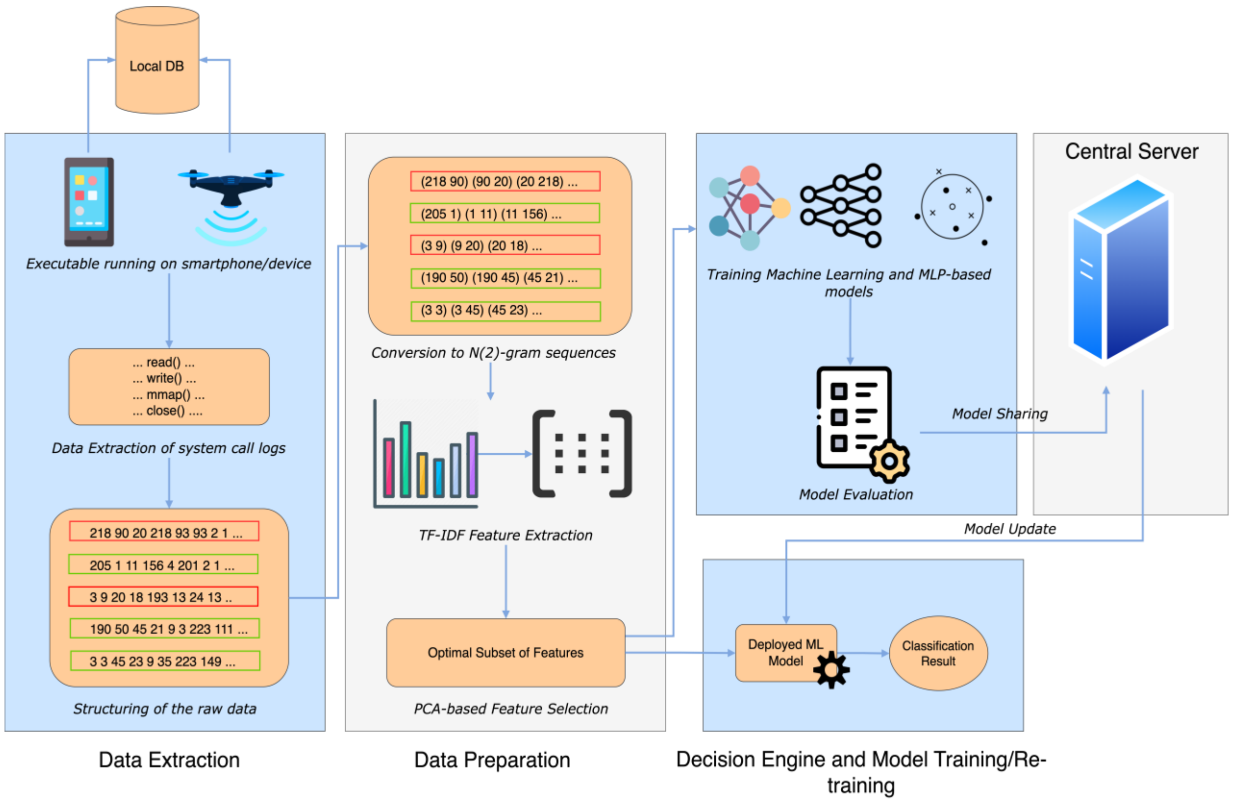


Figure 18 - Detailed architecture of the system call-based HIDS solution

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 is not directly on the device). 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 intrusion 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 is to establish a comparison between the traditional IDS approaches and the federated setting, where the centralized architecture serves 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 to be 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 resulting data is parsed – we extract one system call per row. 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.

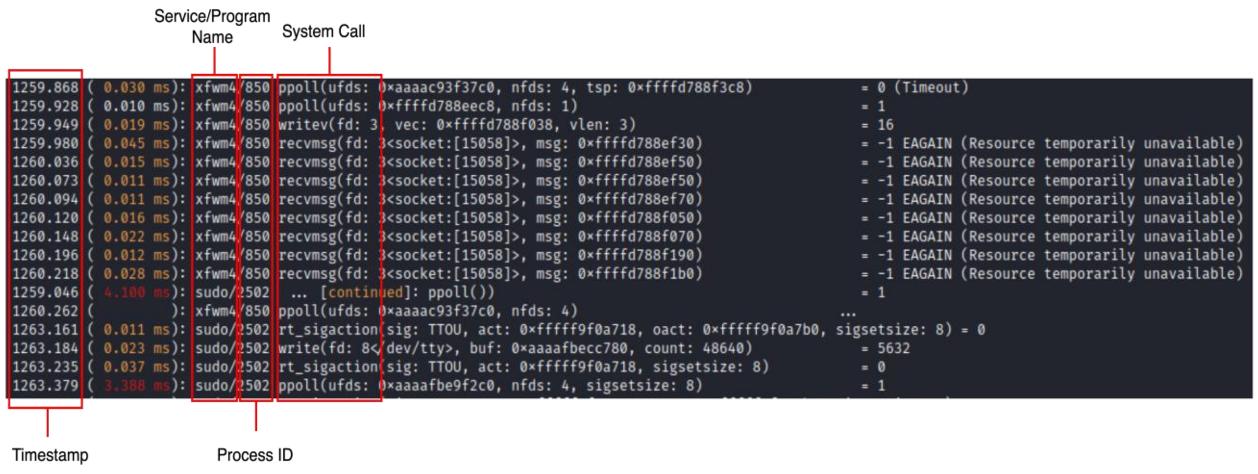
After the tokenization process, the data is sanitized by removing the intersecting rows (sequences) from both normal and attack datasets to maximize the distinction between the two class types for the learning process. A row with a normal sequence is set with a label of 0, whereas a row with an intrusive sequence is set with a label of 1. Having done that, we follow that up with 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. After performing the Feature Extraction step, the most relevant features are selected, the resulting dataset is then divided into a 70/30 split for training and testing, respectively.

3.2.2.3 Produced resources

The following functionalities are the result of the development done on 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[18] of the host device in real time. Figure 19 presents an example of a raw system call log extracted with the *perf* tool. 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.

[18] https://github.com/torvalds/linux/tree/master/tools/perf



```
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 19 - 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., Device Self-Protection or Reputation System).

The developed code written in Python3 is available on the GitLab repository [19].

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 20 illustrates the logical architecture that follows the Device Behaviour Monitoring component, of both internal and external interfaces.

[19] https://gitlab.com/arcadian_iot/device_behaviour_monitoring

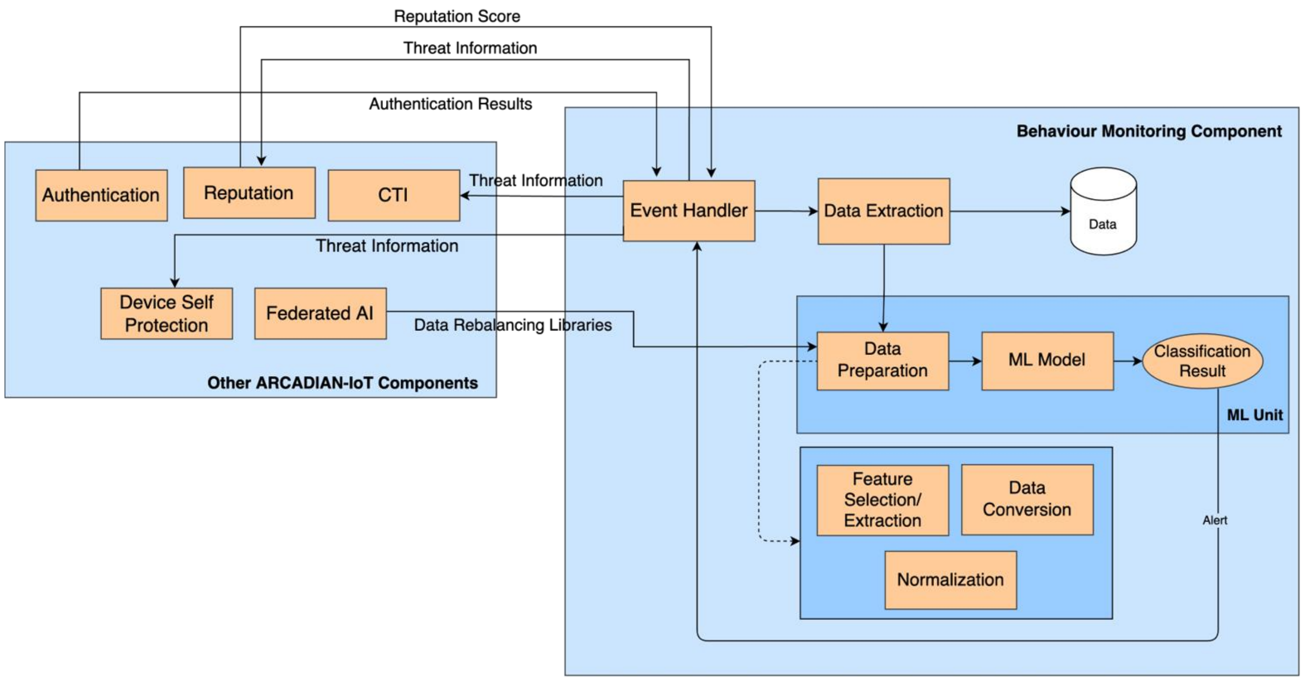


Figure 20 - Behaviour Monitoring component architecture

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

- Data Extraction: Continuously extracts raw data directly from the device pertinent to user activity (e.g., system call traces) with the *perf* Linux based tool for system call log activity (for Linux based devices).
- Detection Module: The data from the Data Preparation Module is forwarded to the detection module, which will consist of ML models that will indicate, given an input, if it is an intrusion or not. If an intrusion is detected from the event traces, then the model will classify what type of intrusion occurred
- 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 will output a classification of either normal or intrusion from the input data
- Detection Module: The data from the Data Preparation Module is forwarded to the detection module, which will consist of ML models that will indicate, given an input, if it is an intrusion or not. If an intrusion is detected from the event traces, then the model will classify what type of intrusion occurred
- Event Handler/Alarm: The resulting classification of the intrusion 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.

3.2.3.2 Sequence diagrams

The sequence diagram in Figure 21 depicts the expected sequence of events in case of a security or privacy incident. The Device behaviour 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.

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.

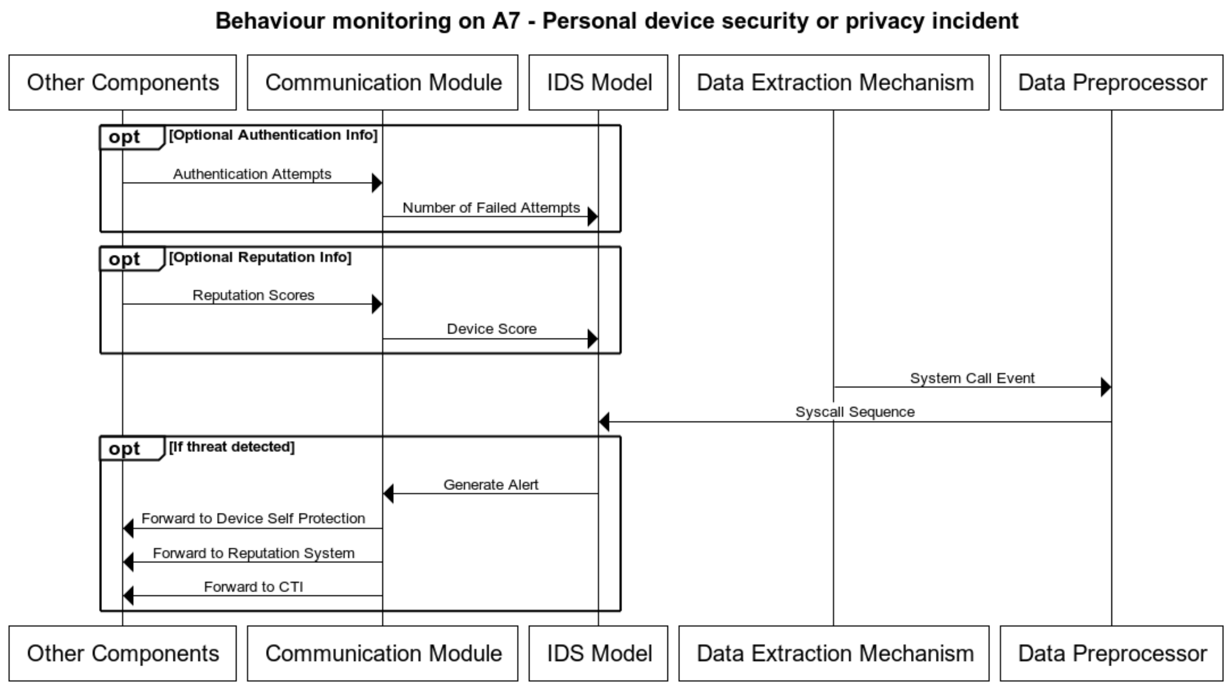

Behaviour monitoring on A7 - Personal device security or privacy incident
Other Components
Communication Module
IDS Model
Data Extraction Mechanism
Data Preprocessor
opt
[Optional Authentication Info]
Authentication Attempts
Number of Failed Attempts
opt
[Optional Reputation Info]
Reputation Scores
Device 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 21 - 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. 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 intrusive behaviour).
- Authentication: Interface used to receive authentication results from the Authentication component (e.g., consecutive authentication attempts)

The following table details the progress on the external interfaces:

Partner	Definition	Channel	Exchange	Status
IPN	Device Self Protection	RabbitMQ	Threat/intrusion	In

		AMQP 0.9.1	detection warnings	progress
UC	Reputation System	RabbitMQ AMQP 0.9.1	Threat/intrusion detection warnings	In progress
RISE	CTI	RabbitMQ AMQP 0.9.1	Threat/intrusion detection warnings	Pending
UC	Reputation System	RabbitMQ AMQP 0.9.1	Reputation updates	In progress
RISE	Federated AI	Direct Code Integration	Data Rebalancing Libraries	In progress
TRU	Authentication	RabbitMQ AMQP 0.9.1	Authentication Updates	Pending

Table 5 - Behaviour Monitoring External Interfaces

The external interfaces that have the status of ‘Pending’ refer to component integration that is relative to the second prototype, and thus, the efforts for integration of the two components have not yet started.

3.2.3.4 Technical solution

3.2.3.4.1 Deployment architecture view

The behaviour monitoring component will enable two main types of deployment. The first type of deployment (Figure 22), 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 (Figure 23), safe for the data extraction subcomponent, which is done on the device, the information is then sent by a safe channel in batches to a cloud instance of the component, which processes the data.

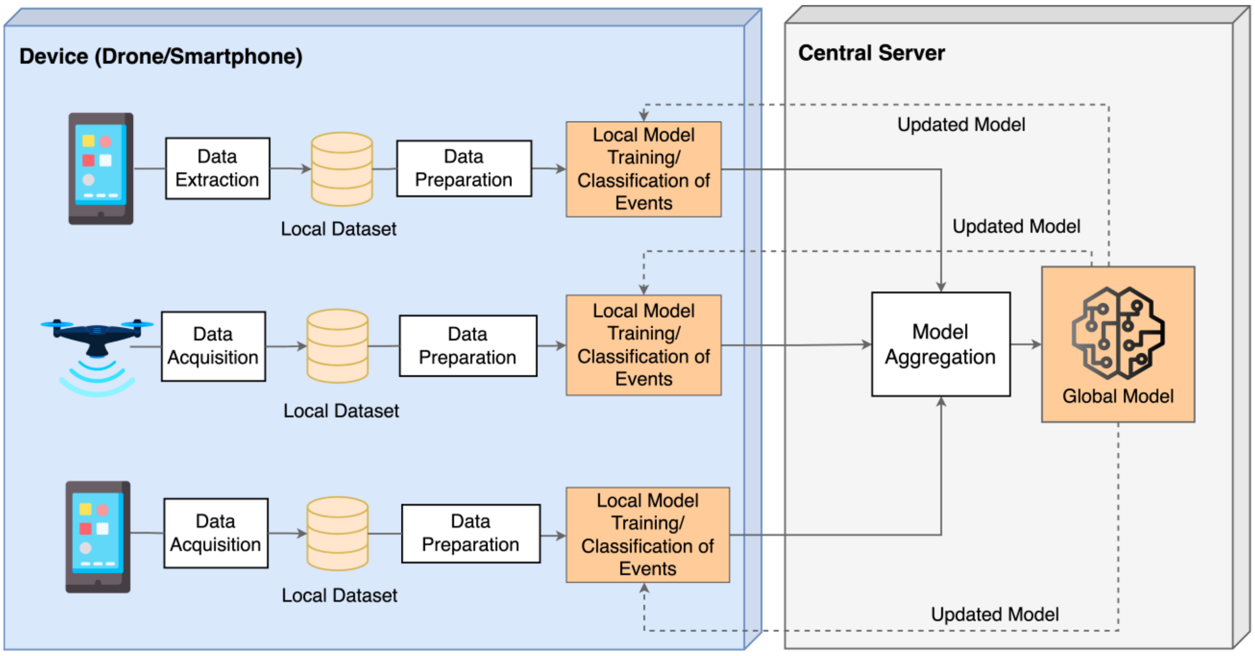


Figure 22 - On device deployment architecture view

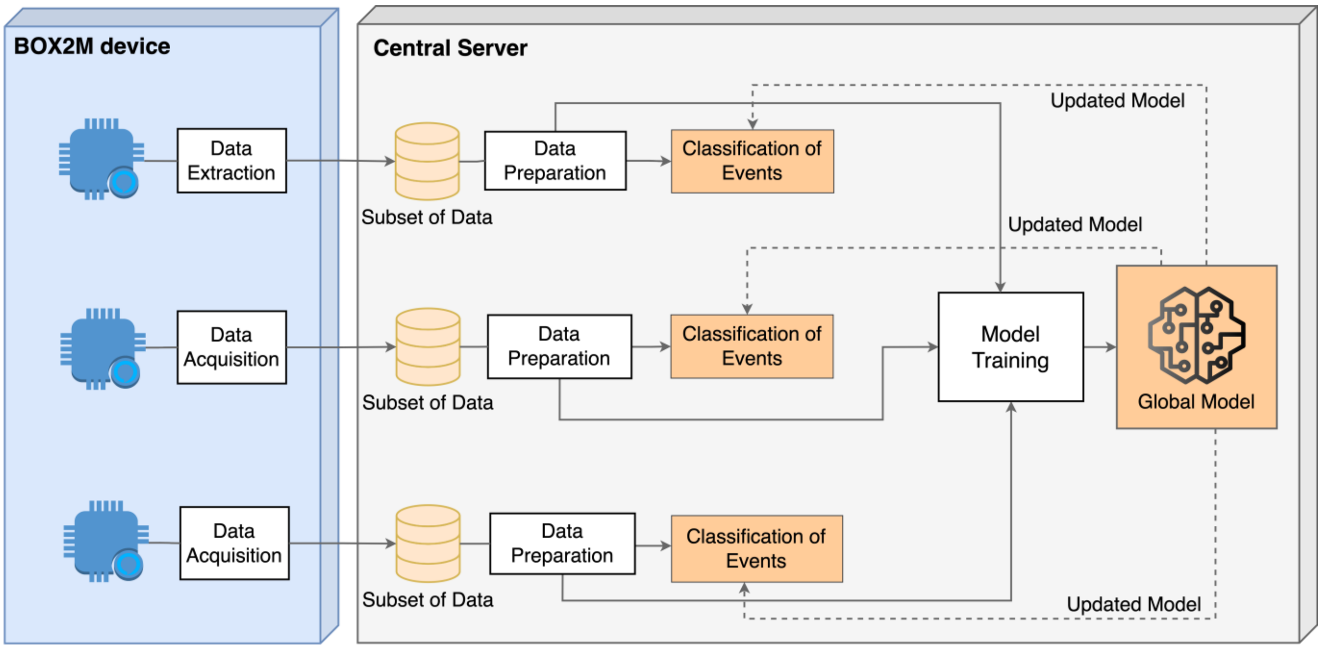


Figure 23 - Non-device deployment architecture view

3.2.3.4.2 *API specification*

The Device Behaviour Monitoring component relies on a RabbitMQ publication/subscription channel to receive and forward information to the rest of the ARCADIAN-IoT framework components. The output of the component is composed of information about the detected intrusions. Table 6 details the output specification of the component.

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)
	device_id	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)
	process_id	string	Process ID associated to the program	ID number of the process (e.g., 9346)
	sender	string	Identification of sender component	<domain> <component> <loc ation> <entity_id> <service>

Table 6 - Device Behaviour Monitoring Output Specification

3.2.3.4.3 Security aspects

It is noteworthy that the device behaviour monitoring seeks to maximise the privacy of the devices that integrate the component at hand. This is achieved by applying Federated Learning mechanisms when a model update is needed, this means that, data that is relative to the device is not shared with outside entities, as only the model parameters of the local device model is shared with a centralised server that performs model aggregation, which keeps data secure and private.

Furthermore, security as it relates to communication with other ARCADIAN-IoT components, the Behaviour Monitoring component uses RabbitMQ with message bus, which is authenticated, and further security layer can be added (with certificate usage is considered for a later version of the component).

3.2.4 Evaluation and results

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, some performance metrics were selected to achieve that. The performance of an IDS can be measured by 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 7 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 7 - 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 done 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 8 shows the metrics obtained with the experimentations done 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 8 - FL Model Training Results

Based on the values obtained, we were able to achieve close results 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.

3.2.5 Future work

Since the first version of the component is done, current work of the Device Behaviour Monitoring is focused on integration aspects with other ARCADIAN-IoT components, namely Device Self-Protection and Federated AI.

The subsequent work will focus on integration with other ARCADIAN-IoT components, such as CTI, Authentication and Reputation Systems. Additionally, some adjustments in the functionalities (e.g., multi-class models) and more experimental activities need to be done.

With the prototyping finished, the component will be integrated in the three domains of the project and validated in the use cases in which the component is involved. Moreover, additional adjustments can be made in the layer of communication with other ARCADIAN-IoT components.

3.3 Cyber Threat Intelligence (RISE)

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 is building 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) generation of 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, access control, and automatization which would be useful in IoT environments. Understanding the quality/reliability and timeliness 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 will 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 (M30)	***Current value (M20)***
# types of threat > 5 # IoC formats > 3	# types of threat = 2 # IoC formats = 2

KPI scope	
Provide ML models for automated IoC clustering	
Measurable Indicator	
Purity, Silhouette Coefficient	
Target value (M30)	***Current value (M20)***
Purity > 70% Silhouette Coefficient > 70%	n/a as the subcomponents relying on ML models are still under development

KPI scope	
Provide ML models for automated IoC ranking	
Measurable Indicator	
NDCG, MAP, Precison@k	
Target value (M30)	***Current value (M20)***
NDCG > 80% MAP > 80% Precison@k > 80%	n/a as the subcomponents relying on ML models are still under development

KPI scope	
Provide ML models for automated Event/Galaxy classification	
Measurable Indicator	
Accuracy, F1-score	
Target value (M30)	***Current value (M20)***
Accuracy > 85% F1 > 0.85	n/a as the subcomponents relying on ML models are still under development

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.

In current reporting period, RISE has focused on launching MISP core and configuring it to meet requirements for different use cases. The main contributions in this task can be summarized as follows:

- Launched and configured MISP in RISE Cyber Range for the future exploit.
- Defined architectural details of the ARCADIAN-IoT CTI component and its subcomponents requirements.
- Implemented a set of extended functionalities for MISP core including automatic event/attribute fetching, updating, and searching.
- Defined the format for lightweight Indicator of Compromise (IoC) which can be used in constraint environment.
- Started the implementation of advanced ML-based functionalities for IoCs processing.

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
- 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.

The MISP platform includes three main roles in MISP in the management process: the general administrator, organization administrator, and publisher. CTI platform setup will include three different instances in three distinct locations controlled by RISE, IPN, and Truphone (TRU). In our current setup, RISE takes the role of general administrator, while IPN and TRU will be granted with organization administrator credentials in a second stage. The general administrator has the highest authority to manage all the collected threat data and is in charge of monitoring how the information is being shared. Organization administrators have the authority to manage organizational threat events.

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

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.).

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 aim to implement 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 been investigating ways 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 packets 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.

RISE has also been working on building up CTI interfaces to parse and aggregate alerts from ARCADIAN-IoT sources and external sources. RISE has implemented a JSON parser to extract information from the alerts of network events provided by Network Flow Monitoring component and convert the input into MISP format. Next, the parser is then linked with the functionalities to push the events to the MISP database. we implemented an aggregator to collect STIXs from the external databases.

Finally, RISE has defined three different ML-based modules to be used for smart IoCs processing and analysis: (i) threat level ranking, (ii) federated IoC classification, and smart IoC sharing. The threat level ranking function automatically ranks the received IoCs or data based on their threat level. The self-labelling task is performed via a DL-based ranking model, and an assigned threat level will be added to the IoCs or corresponding MISP event; The federated IoC classifier integrates the Federated AI component into the CTI and will classify the unlabeled IoCs to the correct event or galaxy, after a federated training process. The smart IoC sharing module can be seen as an IoC recommendation system. This module distributes the IoCs to the subscribers according to their tags/galaxies, and also shares specific IoCs according to levels of interests.

3.3.2.3 Produced resources

Due to the early stage of development, there are no resources currently available to public. The source code of the extended functionalities of the MISP core will be made available once all the evaluations are completed.

3.3.3 Design specification

3.3.3.1 Logical architecture view

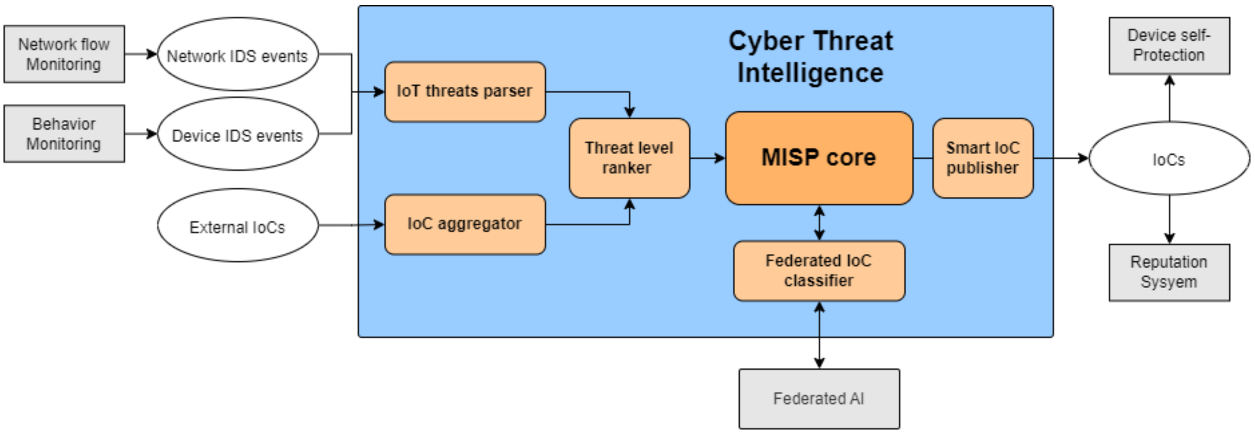


Figure 24 - The architecture of CTI

Figure 24 shows the architecture of CTI which consists of the MISP core, IoT IDS event parsing and formatting, IoC aggregation, and ML model manager. Each subcomponent is in charge of different tasks.

- **MISP core**: It is the CTI engine that provides primary features of threat data gathering and sharing. It also includes extensive functionalities for automation and management.
- **IoT threats parser**: It is the subcomponent that aggregates and analyses IDS events from devices in the IoT network. Because traditional standards for cyber threat information (e.g., STIX, TAXII,[2] etc.) requires formats which are too heavy for resource-constrained IoT devices, we aim to develop a lightweight format and protocol that are suitable for communication of IoT-specific IoCs.
- **IoC aggregator**: It is responsible for the aggregation and pre-processing of existing IoT-specific vulnerability databases. It transfers the vulnerable information into IoC format and import them to MISP.
- **Subcomponents with ML functionalities**: The CTI includes three ML-based modules: threat level ranker, Federated IoC classifier, and smart IoC publisher. Each module is in charge of a specific ML-based function for IoC processing and analysis. The threat level ranking module automatically ranks the received IoCs or data based on their threat level. It is a self-labelling task, and an assigned threat level will be added to the IoCs or corresponding MISP event; Federated-Integrator is responsible for integration between CTI and Federated AI component. A ML model for event/galaxy classification is trained in federated learning setup. This model helps to classify the unlabelled IoCs to the correct event or galaxy. Finally, the Smart IoC works like an IoC recommender system and it distributes the IoCs to the subscribers according to their tags/galaxies.

3.3.3.2 Sequence diagrams

Figure 25 describes the sequence diagram of the CTI system which includes the functions for IoC threat parsing, IoC aggregation, threat level ranking, federated IoC classification, and smart IoC sharing. 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 re-format the data in MISP format and will transfer the IoC to the MISP core. If there is no threat level associated with the IoC, a threat level is automatically assigned by the ML-based threat level ranker before being transferred to the MISP core. If the event is unlabelled, i.e. no event type and galaxy is associated with the IoC, a label is automatically assigned by the federated classifier before being transferred to the

MISP core. The analyst may exploit MISP core and interface for threat analysis and visualization. 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.

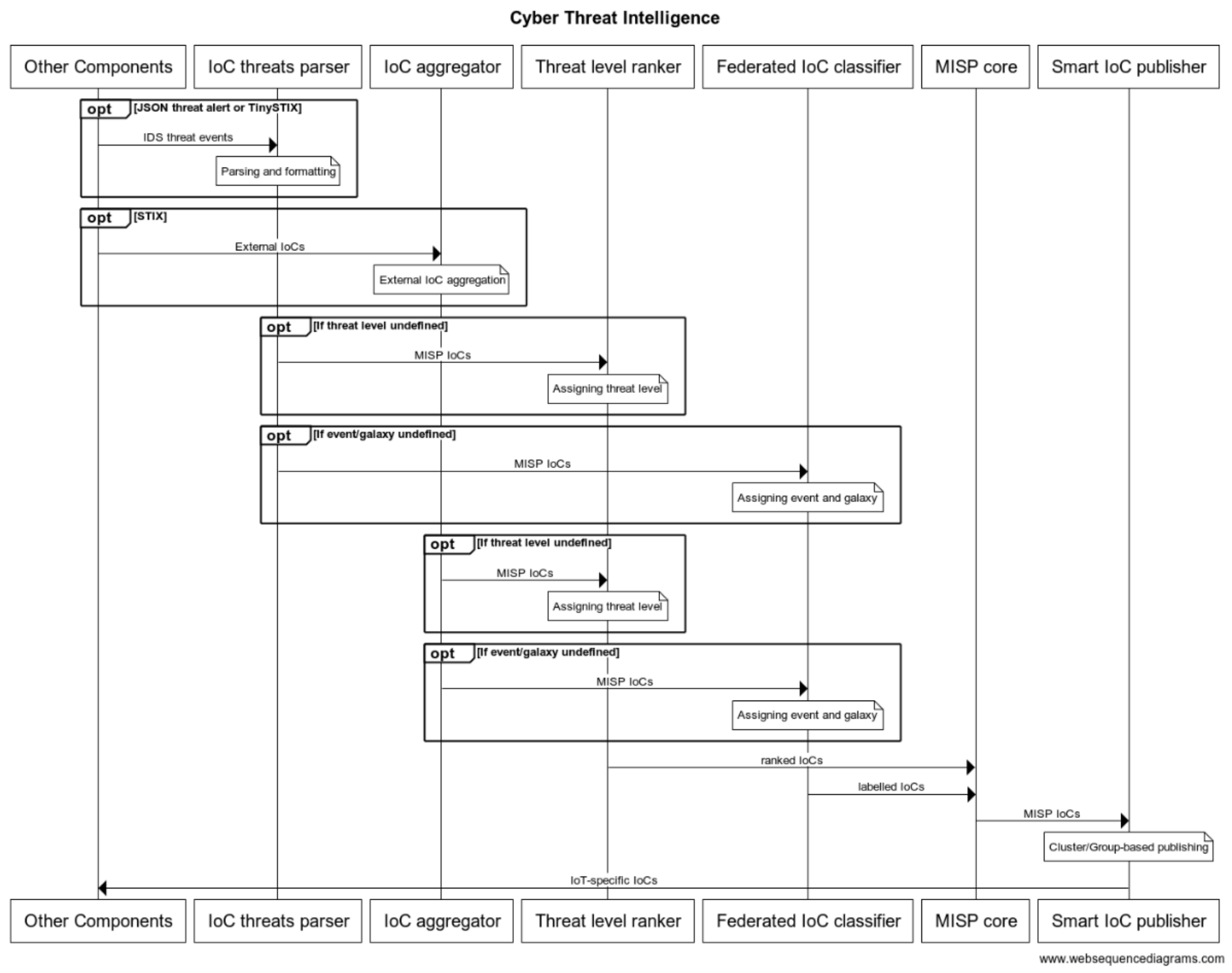


Figure 25 - Sequence diagram

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 will use 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 will send alerts as results of the attack detection. It will use a protocol to send the alerts to a publication/subscription middleware via RabbitMQ. CTI system will subscribe to such alerts and process the alerts for further analysis and aggregation.
- Federated AI: CTI will have 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.

The following table details the progress on the external interfaces:

Partner	Definition	Channel	Exchange	Status
UWS	Network Flow Monitoring	RabbitMQ AMQP 0.9.1	Threat/intrusion detection warnings	In progress
IPN	Behaviour Monitoring	RabbitMQ AMQP 0.9.1	Threat/intrusion detection warnings	Pending
IPN	Reputation System	RabbitMQ AMQP 0.9.1	Indicators of Compromise	Pending
IPN	IoT Device Self-protection	RabbitMQ AMQP 0.9.1	Indicators of Compromise	Pending
RISE	Federated AI	Direct Code Integration	Communication-efficient and robust aggregation rule Libraries	Pending

Table 9 - The external interfaces of the CTI component

The external interfaces that have the status of ‘Pending’ refer to component integration that is relative to the second prototype, and thus, the efforts for integration of the two components have not yet started.

3.3.3.4 Technical solution

3.3.3.4.1 API specification

As the development of the CTI system and its interface is ongoing, API specifications are not available yet at the time of writing.

3.3.4 Evaluation and results

The current prototype is regarded as a preliminary step towards the integration with other components in the ARCADIAN-IoT framework. The evaluation metrics are aligned with the purpose of KPIs and aim at defining which and how many IoT-specific threats the CTI platform can handle, how many IoT protocols/technologies can be supported, and how many organization/stakeholders contributes to the information sharing via the platform. At the time of writing this deliverable no measurable results are available yet. We are currently analysing the impact of using tinySTIX and tinyTAXII for IoC sharing in IoT environments. A deep performance evaluation will be conducted to evaluate the ML-based subcomponents as soon as the implementation and integration in the CTI system is completed.

3.3.5 Future work

Future work will focus on the following: (i) complete the implementation of the IoC aggregator for external resources, (ii) complete the implementation of the ML-based modules for advanced IoC processing and analysis, (iii) integration and deployment of the Federated AI component and its subcomponents into the CTI system, (iv) performance evaluation.

TRU will join the efforts of this component by bringing into the consortium the expert knowledge of their CSIRT (Computer Security Incident Response Team), TRU and IPN will have a node of the CTI platform running on their premises to monitor and analyse any relevant threats.

3.4 Network Self-protection (UWS)

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 is being 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 execute 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, stop de attack and restore 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 3.4.10. 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.	
Target value (M30)	***Current value (M20)***
>= 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.	
Target value (M30)	***Current value (M20)***
> 95% service recovery Towards 0 % human intervention	> 95% service recovery Towards 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	
Target value (M30)	***Current value (M20)***
Up to 10^6 malicious flows Support to work up to 10 Gbps	Up to 10^6 malicious flows Support to work up to 10 Gbps

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 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 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 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

20 https://www.dpdk.org/

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

22 https://www.openvswitch.org/

23 https://www.openstack.org/

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

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 with 1) and 2) 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 3) 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 extract 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 architectural design of the 5G IoT classifier is provided in Figure 26, 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.

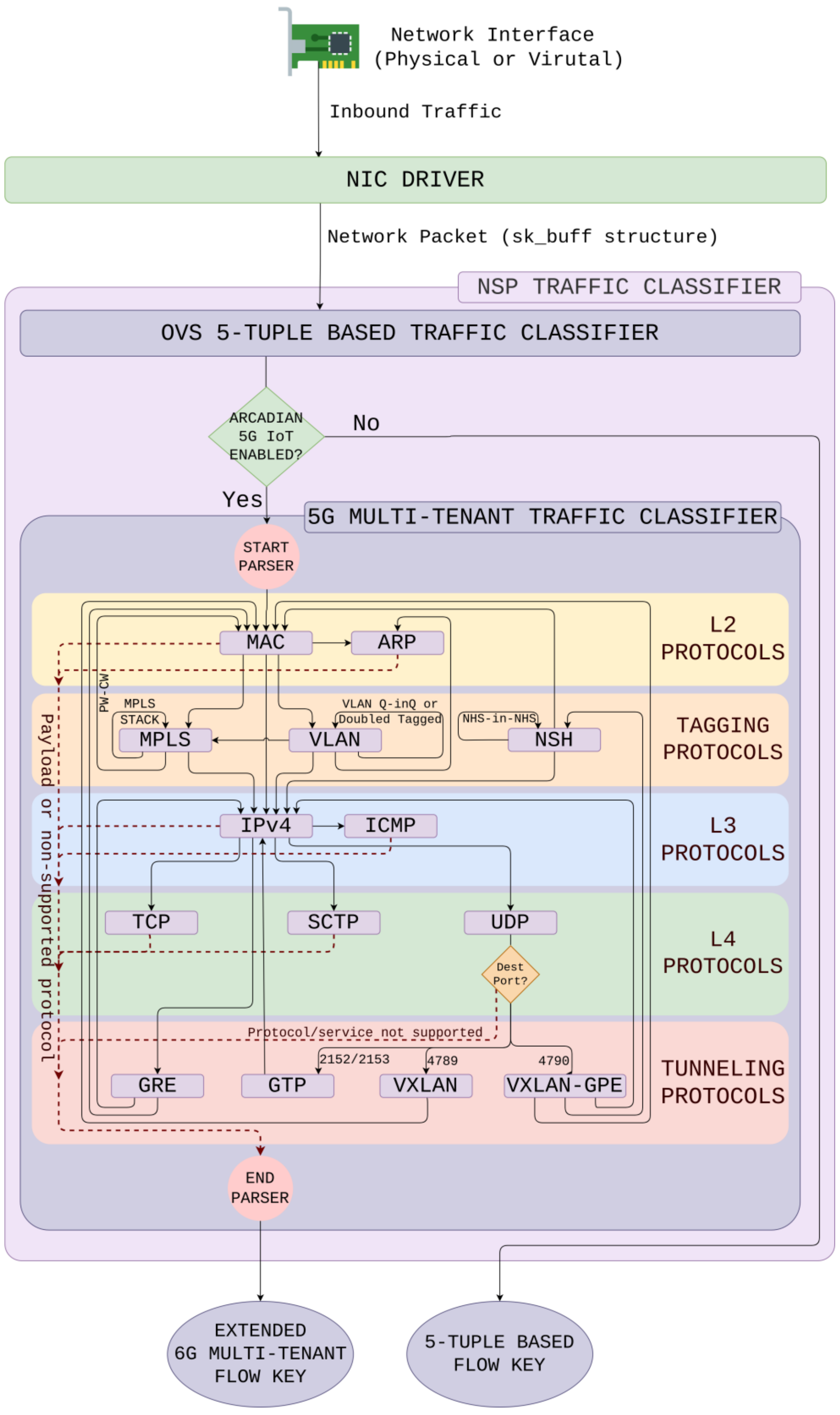


Figure 26 - Overview of the 5G IoT integrated in the Network Self-Protection

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 10 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	Destination IP address 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 10 - Proposed new Open Flow fields

It is worth to highlighting that the protocol header field matching the tunnel ID depends on each encapsulation protocol context as indicated in Table 11. 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	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 11 - Parsing of encapsulation, tunneling and tagging protocols: tunnel type values and

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 27, 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

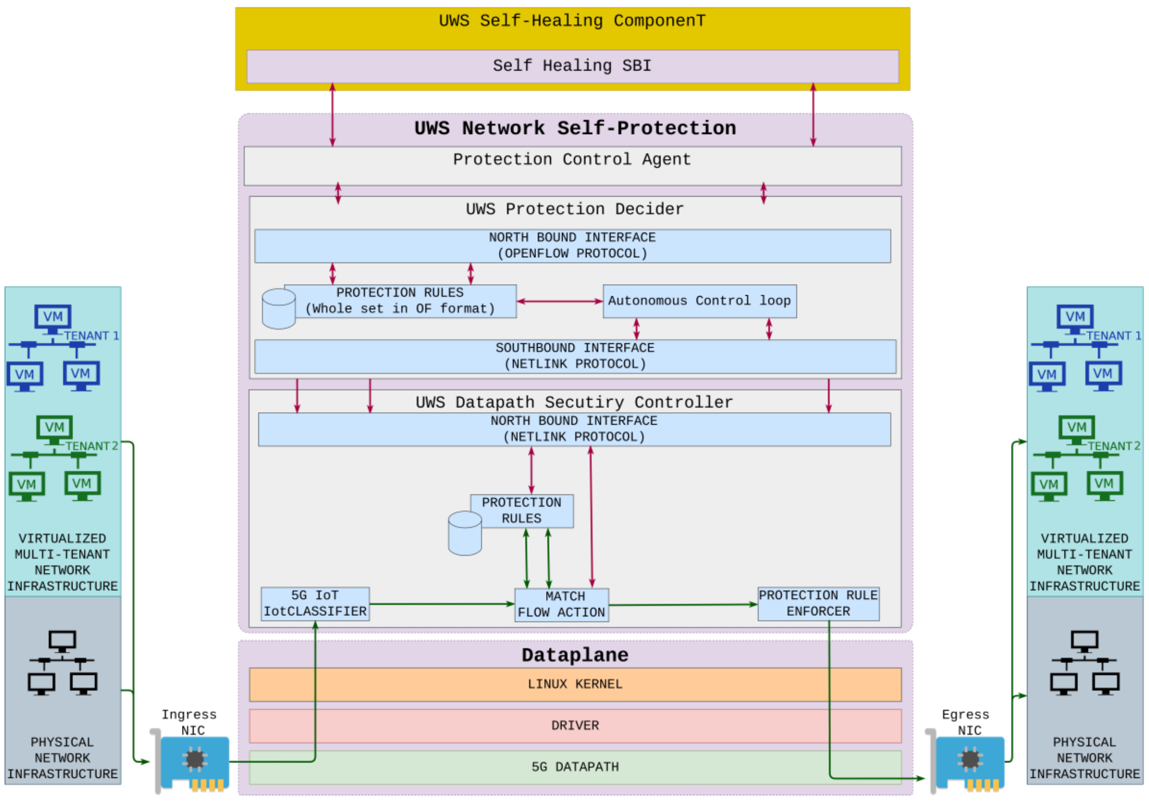

UWS Self-Healing ComponenT
Self Healing SBI
UWS Network Self-Protection
Protection Control Agent
UWS Protection Decider
NORTH BOUND INTERFACE (OPENFLOW PROTOCOL)
PROTECTION RULES (Whole set in OF format)
Autonomous Control loop
SOUTHBOUND INTERFACE (NETLINK PROTOCOL)
UWS Datapath Secutiry Controller
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 27 - 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 into the PD will be enforced in the DSC by creating an autonomous control loop to perform self-optimization of the protectional capabilities and thus achieving large scalability, really needed to deal with security on IoT networks. The PD is continuously monitoring the behaviour of the DSC 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 DSC 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 a 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 28 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 28). This alert is consumed and processed by the Network Self-Healing component (2 in Figure 28). The NSH, based on the network topological and resources information gathered, takes a decision on how to stop the attack: In 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 28). 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.

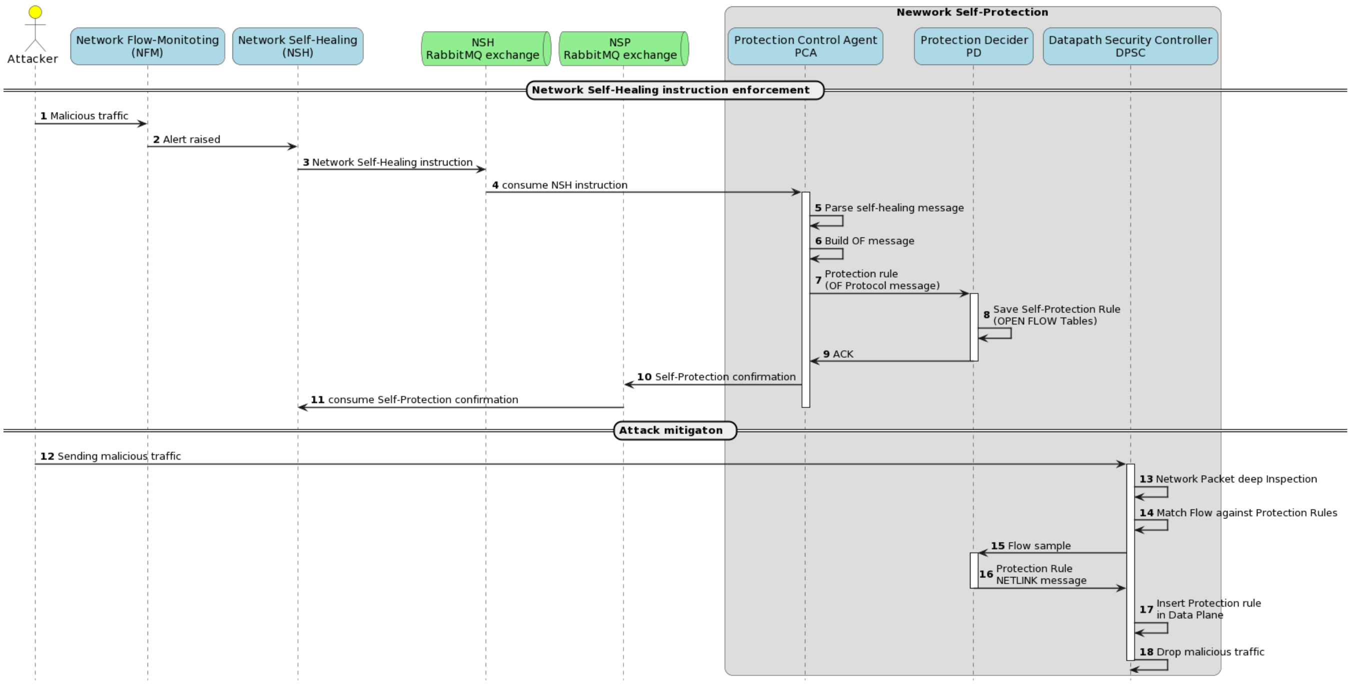

Attacker
Network Flow-Monitoting (NFM)
Network Self-Healing (NSH)
NSH RabbitMQ exchange
NSP RabbitMQ exchange
Newwork Self-Protection
Protection Control Agent PCA
Protection Decider PD
Datapath Security Controller DPSC
Network Self-Healing instruction enforcement
1 Malicious traffic
2 Alert raised
3 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 28 - Network Self-Protection architectural overview

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 28). 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 28), 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 28). 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 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 28). The PD sends to the DPSC the self-protection rule to be inserted in the data plane (16 in Figure 28). 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 28), all harmful traffic matching the rule will be dropped, resulting in stopping the attack at the desired point in the network (18 in Figure 28).

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	Status
UWS	Network Self-Healing	RabbitMQ AMQP 0.9.1	Network Self-Healing Instructions	Done
UWS	Network Flow Monitoring	RabbitMQ AMQP 0.9.1	Network Self-Protection Confirmations	Done
UC	Reputation System	RabbitMQ AMQP 0.9.1	Network Self-Protection Confirmations	Pending

3.4.3.4 Technical solution

3.4.3.4.1 Deployment architecture view

Figure 29 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.

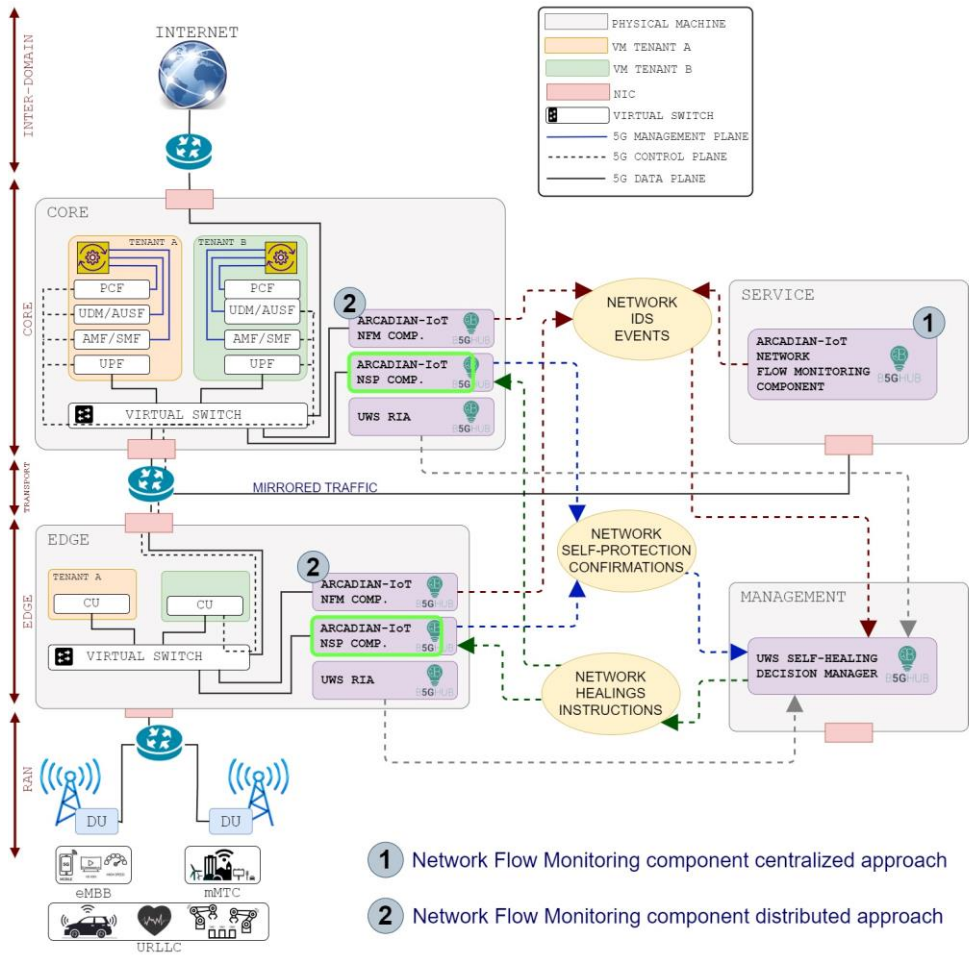


Figure 29 - Network Self-Protection architectural overview

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 -IoT 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

At this stage of the project, a first functional prototype of the network Self-Protection component is already developed. The ongoing efforts are focused on completing the integration with the rest of the architectural components of the ARCADIAN-IoT framework. Following the prototyping, the overall behaviour of the Network Self-protection component will be empirically tested as a preliminary step towards the integration with the rest of the architectural components composing the Horizontal Planes of the ARCADIAN-IoT framework. The tests will be carried out in the cloud data centre at the UWS premises and will be focused on two aspects: functional validation and overall performance.

Regarding functionality, it will be empirically verified that the component is able to effectively enforce in the data plane the set of healing actions sent by the Network Self-healing component. The purpose of these actions is to stop malicious traffic coming from known DDoS attacks previously detected by the Network Flow Monitoring component.

Regarding performance, the following features will be evaluated to demonstrate the suitability of the component for its integration into a 5G IoT architecture such as the one defined in ARCADIAN-IoT:

- **Scalability:** Maximum number of IoT devices traffic simultaneously managed.
- **Bandwidth:** Maximum data rate that the component is able to deal with.
- **Classifier overhead:** Comparative of the extra overhead introduced when processing different profiles of complex traffic (overlay networks with different number of nested headers).
- **Delay:** Average time to process each packet. That is, the time required to classify the packet and take a decision based on the current protection rule set enforced in the data plane. The default actions may be:
    - To drop traffic if it is identified as malicious traffic.
    - To process the packets according to the routing tables in the case of traffic labelled as legitimate.
    - Any other kind of actions could be considered for the harmful traffic such as mirroring, redirecting to a Honeynet, sending to the SDN controller, etc.

Concerning the type of experiments executed, the component will be stressed against several challenging scenarios with different traffic profiles ranging several parameters:

- Packet size
- Number of IoT devices sending traffic
- Tx Bandwidth per IoT device
- Number and type of headers for tunnelling and overlay networks.

3.4.5 Future work

The UWS team is currently working on the integration between the three components that belong to the cognitive self-protection network security loop. The development of the first version of the NSP component prototype is achieved and is being integrated with the rest of the ARCADIAN-IoT framework components

After prototyping and as a last step in the development of the component, the prototype will be empirically validated and evaluated (as described in previous Subsection). Hence, an empirical validation and evaluation of the overall performance of this component will be reported in deliverable D3.3. Finally, this component will be integrated in the ARCADIAN-IoT architecture, where it will interact with different architectural components. The major integration work will be undertaken with the Network Self-healing and the Network Flow Monitoring components. These three components will cooperate to provide an autonomous cognitive self-healing loop in the network: The Network Flow Monitoring detects the attack and raises an alert; the Network Self-healing decides what action to take to stop and mitigate the attack and the Network Self-protection enforces the action in the data plane.

3.5 IoT Device Self-protection (IPN)

3.5.1 Overview

3.5.1.1 Description

In order to protect IoT devices from possible threats and respond to intrusions and anomalies detected in them by other components of the ARCADIAN-IoT project, a tool will be implemented that must comply with the aforementioned. The IoT Device Self-Protection component enforces protection policies and rules at the device level. These policies include disabling malicious applications and preventing them to run or access sensitive data 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 also relies on information derived from other ARCADIAN-IoT components such as Behaviour Monitoring, Cyber Threat Intelligence (CTI) or Reputation System. These will provide information on indicators of compromise and reputation levels, which, when combined with different levels of risk, make it possible to create a greater variety of policies and thus better protect IoT devices.

After analysing the messages that arrive at the component, the application of policies to vulnerable devices could be suggested. Also, this information could be communicated to Self-Recovery.

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 (M30)	***Current value (M20)***
Support at least 3 policy enforcement methods (e.g., risk levels, reputation information, threat or indicators of	Supporting 1 policy enforcement method (i.e., threat information for intrusion detection system).

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 (M30)	***Current value (M20)***
Support at least 2 Operating Systems and 2 device types	Supporting 1 Operating System (Linux) and 1 device type (drone).

3.5.2 Technology research

This section gives background information on the problem, highlighting research that has been done concerning 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 device management. (https://www.appsealing.com/owasp-iot-top-10/)

For this reason, it is essential to have a security layer capable of reducing the possibility of attacks and intrusions on IoT devices. Self-protection mechanisms are designed to increase runtime software system security. By monitoring the status of a system and its execution environment, making decisions based on the observed state, and implementing enhanced security measures that are appropriate for the present threat, they are able to automatically prevent and react to security risks. The advantage of these mechanisms is that they allow the protection of IoT devices without any human intervention.

3.5.2.2 Research findings and achievements

During the research period, we sought to find self-protection solutions similar to those intended to be implemented in the ARCADIAN-IoT project and which, at the same time, would bring innovation to the component to be implemented. The first step was to find an architecture that best fits the purpose of this component. In this sense, the MAPE-K architecture stands out, which divides the process of a system into the following steps:

- 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; it also 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.

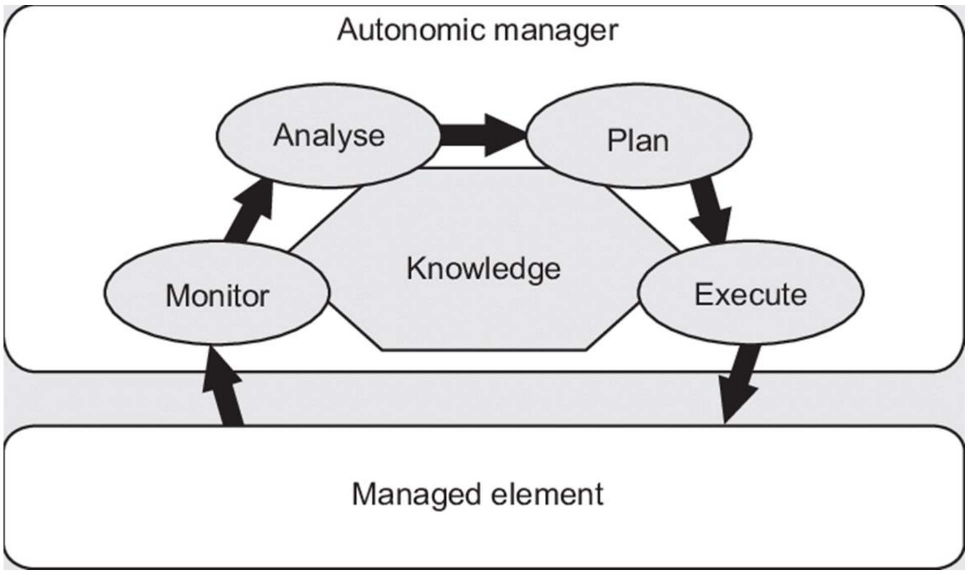


Figure 30 - MAPE-K Architecture Representation[25]

This architecture fits the logical architecture for the IoT Device Self Protection component. The Monitor corresponds to the sub-component responsible for listening to new alerts from other ARCADIAN-IoT components. The Analyzer and Planner blocks correspond to sub-components that deal with data processing, finding the best response to a suspicious state change. The Executer block corresponds to an external interface to the component, which will make the necessary changes recommended by IoT Device Self-Protection. Finally, the Knowledge Base contains the policies that must be applied to each of the threats that can be detected by other components of ARCADIAN-IoT.

During the research process, we tried to find alternatives for data analysis beyond the analysis of the data provided by the components that feed the IoT Device Self-Protection. One of the approaches found is related to the response to an action by risk levels. Take a simple example into account: A failed attempt to log in to an application can be considered a low-risk action. However, if the number of attempts is much higher, the risk of suspicious activity is higher too, and other protection measures may be taken. If in the first example this failed attempt can be ignored, in the second case, it could mean a brute force attack, and measures should be taken (such as requesting a second authentication factor). With this approach it is possible to cover more actions and thus add layers of security to the devices.

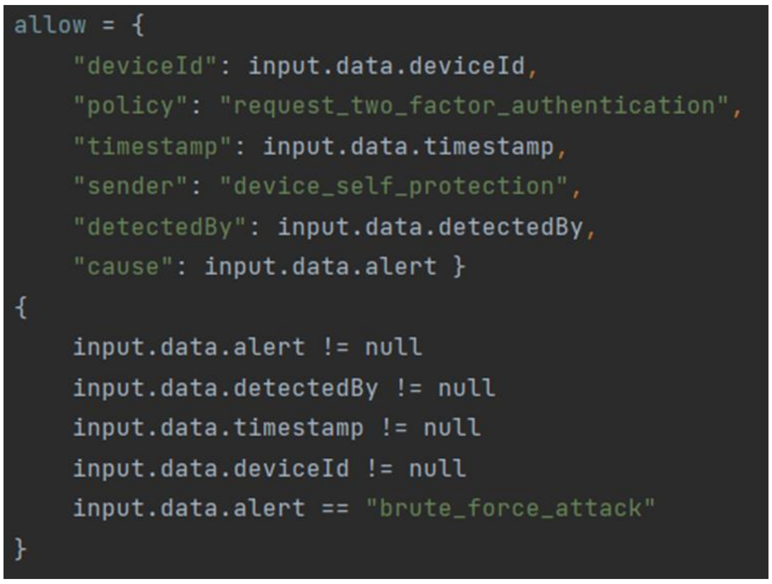
Finally, another question regarding IoT Device Self-Protection was related to how to save device protection policies. The hypothesis of using a database was immediately set aside, as it was a too complex solution if this component had to be installed on devices. In addition, its access would be slower, leading to a loss of component's performance. The solution found goes through the OPA policy engine. With this tool, it is possible to create new policies and save them in a file in .rego format. In this way, in addition to ensuring the performance of the component, writing and reading the policies is simplified.

[25] (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])

3.5.2.3 Produced resources

The IoT Device Self-Protection component's development produced the following functionalities:

- Supports communication with ARCADIAN-IoT messages bus (via RabbitMQ). This functionality allows the device self-protection to receive threat alerts from components such as Device Behaviour Monitoring (P1), reputation scores from the Reputation System (P1) and Indicators of Compromise from the CTI (planned for P2). 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 standardise formats.
- 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 a number of different self-protection policies in an efficient and structured manner. The policies may include an enumeration of actions to perform in order 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.
- Finally, the device self-protection component supports (in the use cases where this is allowed) 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 in order to apply such policy). In the cases where automatic rule enforcement is not available, this component sends the recommended actions to the system manager or administrator.



```
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 31 - Policy structure example of Device Self Protection

The developed code written in Python3 is available on the GitLab repository https://gitlab.com/arcadian_iot/device_self_protection.

3.5.3 Design specification

The logical architecture view is taken into account 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 32 illustrates the logical architecture that follows the IoT Device Self-Protection component of both internal and external interfaces.

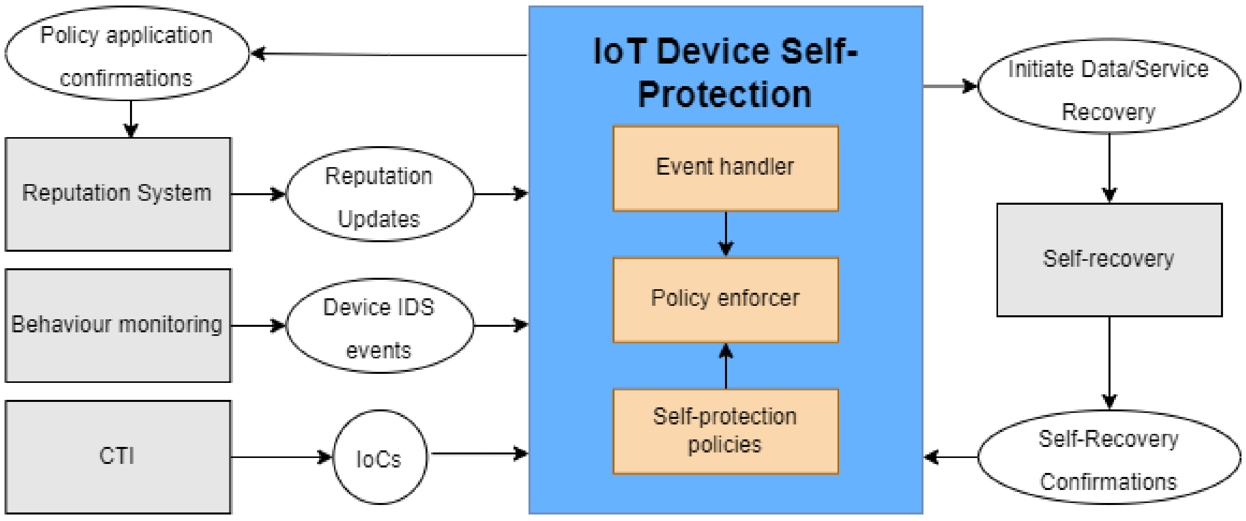


Figure 32 - Device Self-Protection component architecture

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 is in charge of applying/enforcing specific actions on the device (e.g., revoking permissions and closing active connections). Based on the incoming events info, the Policy enforcer probes the self-protection policies to apply the most appropriate actions according to the selected option.
- Self-protection policies: This is a .rego file format with the specification of self-protection policies. This sub-component serves as a knowledge base for the policy enforcer.

3.5.3.2 Sequence diagrams

The sequence diagram in Figure 33 reports the expected behaviour of the IoT Device Self Protection. The first interaction is always from an ARCADIAN-IoT component. The IoT Device Self-Protection could receive intrusion alerts, reputation information or compromise indicators, which the Event Handler will be responsible for processing and translating for the policy enforcer. This information is marked as optional because IoT Device Self Protection is dependent on external data, which will only arrive if some abnormal behaviour is detected. The received message will be analysed in the Self-Protection Policy registry and will suggest which actions to take for a given situation. The Policy enforcer sub-component will be responsible also for recommending an action to be applied to the devices. If a service or an IoT device needs to be restarted, then the Event Handler must forward a message to the Self-Recovery component with all required information.

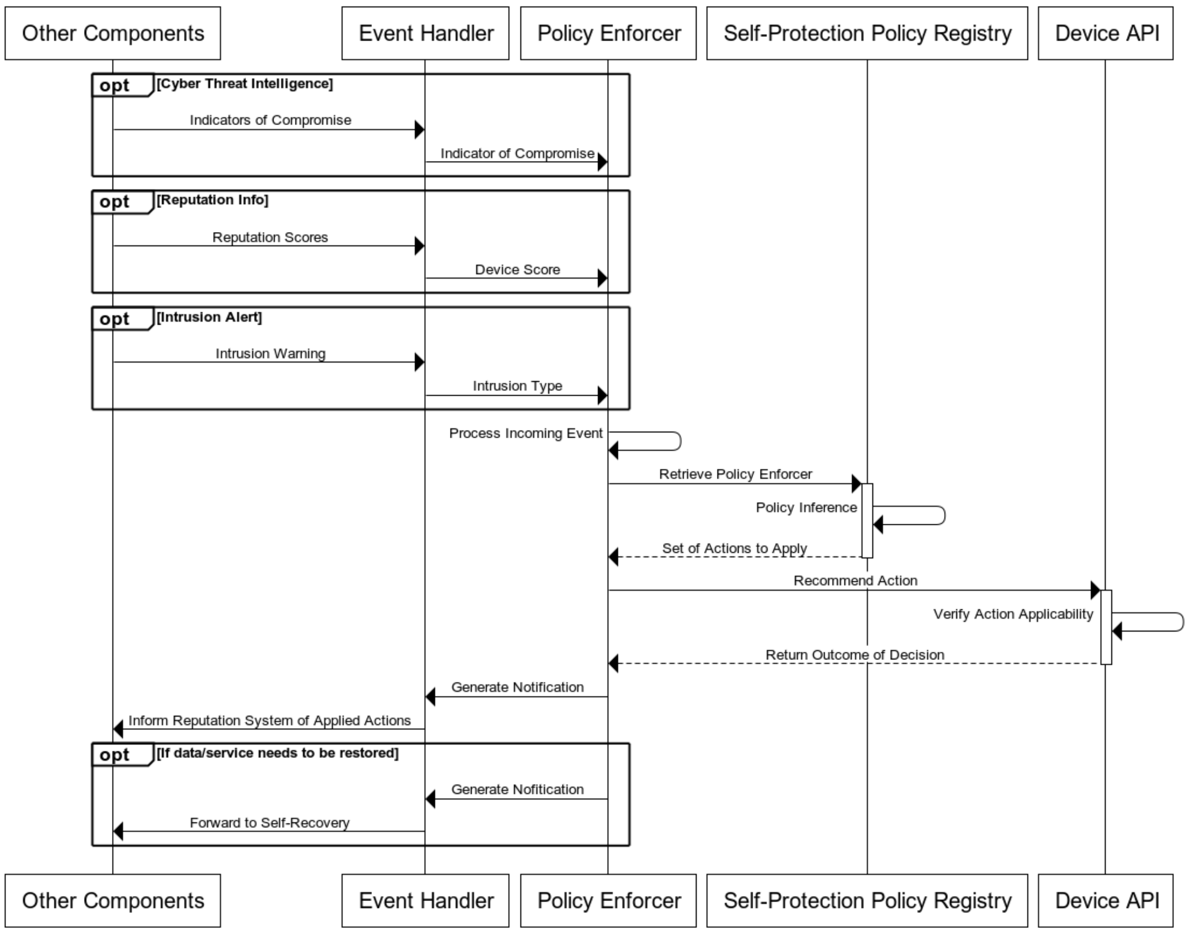


Figure 33 - Expected behaviour of the IoT Device Self Protection

3.5.3.3 Interface description

The external interfaces are the following:

- Behaviour monitoring: Interface used to receive threat/intrusion detection warnings from the Behaviour Monitoring component.
- CTI: Interface designed to receive indicators of compromise or external threat/intrusion-related information from the Cyber Threat Intelligence component.
- Reputation System: Interface designed to receive devices' reputation updates from the Reputation System component.
- Self-Recovery: Interface used to send information to the Self-Recovery component to initiate data or service recovery. It's also used to receive Self-Recovery confirmations about the state of IoT devices.

The following table details the current state of integration with each of the external interfaces:

Partner	Definition	Channel	Exchange	Status
IPN	Behaviour Monitoring	RabbitMQ AMQP 0.9.1	Threat/intrusion detection warnings	In progress
UC	Reputation System	RabbitMQ AMQP 0.9.1	Reputation updates	In progress
RISE	Cyber Threat Intelligence	RabbitMQ	Indicators of	Pending

		AMQP 0.9.1	Compromise	
XLAB	Self-Recovery	RabbitMQ AMQP 0.9.1	Service Recovery and Self-Recovery confirmations	Pending

Table 12 - Device Self-Protection External Interfaces

For prototype one, the main objective was to integrate with the Behaviour Monitoring component. Therefore, the status pending for Cyber Threat Intelligence and Self-Recovery is justified by the lack of need to have this integration ready in the short term. Concerning the integration with the Reputation System component, the process was advanced and this planning has already started.

3.5.3.4 Technical solution

3.5.3.4.1 API specification

A RabbitMQ publication/subscription channel is used by the IoT Device Self-Protection component to receive data from other ARCADIAN-IoT components and transmit data to Reputation System component. The component's output specification is detailed in the next table.

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)
	device_id	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)
	process_id	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>

Table 13 - 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 but actively and passively application/suggesting protection measures in case of intrusions.

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 will not affect its flight control mechanisms.

Additionally, the IoT Device Self-Protection component employs authenticated RabbitMQ with a message bus for connection with other ARCADIAN-IoT components. For a later version, an additional security layer can be implemented (with certificate usage).

3.5.4 Evaluation and results

The first tests carried out focused on the IoT Device Self-Protection sub-components (Event

Handler, Policy Enforcer and Self-Protection Policies). For this, the alert generator that had been developed had particular importance. Taking into account the messages generated, the component was always able to return the expected result for the detected threat/intrusion.

Additionally, it was also possible to implement a functional flow. Starting from reading the messages coming from the message bus, going through the definition and storage of policies and ending with the enforcement of the specific policy for the anomalous event detected on the device.

These positive results give confidence in the robustness of the component. In addition, they facilitate the integration that will be made with the remaining ARCADIAN-IoT components because the alert generator can be adapted to generate messages with the same format as those that will be received in the final version of the project.

Integration with the Behaviour Monitoring component planned for M20 meets what was defined in the KPIs and opens up good prospects for their fulfilment at the end of the project.

3.5.5 Future work

The IoT Device Self-Protection is now integrated with Behaviour Monitoring, as the component's initial version is complete. Integration with other ARCADIAN-IoT components like CTI, Reputation Systems, and Self-Recovery will be the main emphasis of the upcoming work. In addition, some changes to the policies are required.

Once the prototyping is complete, the component will be integrated into the project's domains and tested in the use cases in which it as impact. The layer of communication with other ARCADIAN-IoT components can also be adjusted further.

3.6 Network Self-healing (UWS)

3.6.1 Overview

3.6.1.1 Description

Within the scope of the ARCADIAN-IoT project, UWS is developing 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.	
Benchmarking	
Please provide the values for the indicator obtained by SOTA solutions	
Target value (M30)	***Current value (M20)***

>= 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) 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.	
Measurable Indicator	
Number of network segments supported to get the metadata information.	
Benchmarking	
Please provide the values for the indicator obtained by SOTA solutions	
Target value (M30)	***Current value (M20)***
>= 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.	
Benchmarking	
Please provide the values for the indicator obtained by SOTA solutions	
Target value (M30)	***Current value (M20)***
> 95% service recovery Towards 0 % human intervention	> 95% service recovery Towards 0 % human intervention

3.6.2 Technology research

3.6.2.1 Background

The research carried out by UWS until this point 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 enforcing of rules to heal and protect the network if there is a cyber-attack. 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

At this step we have achieved the architecture of 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 proposed architecture allows the Network Self-healing component to be integrated 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 SHDM and the UWS RIA. RIA has been designed and a first 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 34) such as lldpcli, ispci, iproute2, lshw, brctl and ovs-vsctl 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.

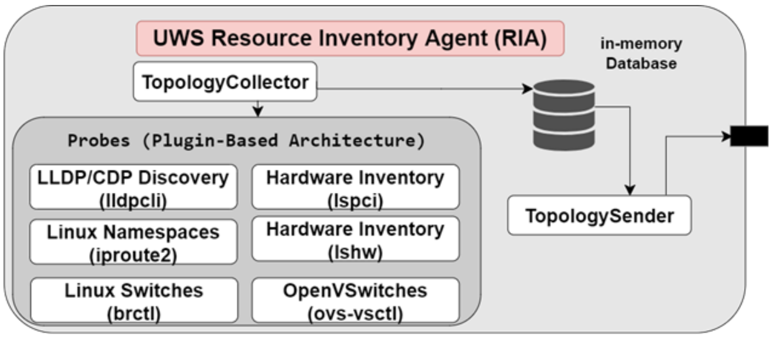

UWS Resource Inventory Agent (RIA)
in-memory Database
TopologyCollector
Probes (Plugin-Based Architecture)
LLDP/CDP Discovery (lldpcli)
Hardware Inventory (lspci)
Linux Namespaces (iproute2)
Hardware Inventory (lshw)
Linux Switches (brctl)
OpenVSwitches (ovs-vsctl)
TopologySender

Figure 34 - 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

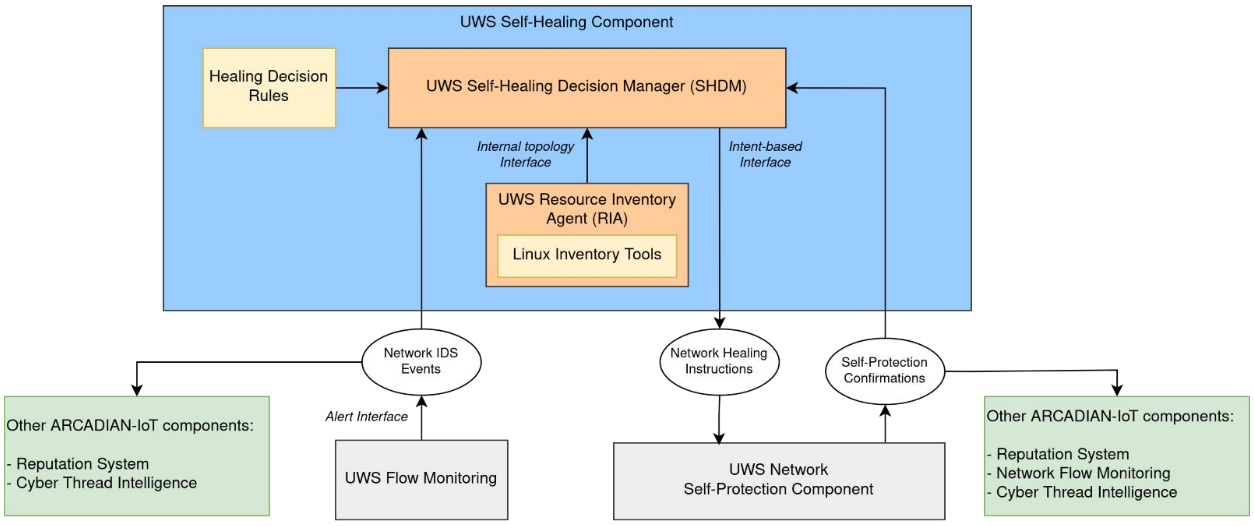

UWS Self-Healing Component
Healing Decision Rules
UWS Self-Healing Decision Manager (SHDM)
Internal topology Interface
Intent-based Interface
UWS Resource Inventory Agent (RIA)
Linux Inventory Tools
Network IDS Events
Network Healing Instructions
Self-Protection Confirmations
Alert Interface
Other ARCADIAN-IoT 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 35 - 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 35):

- 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 aspect 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 36 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 Resource Inventory Agent (RIA) to Self-Healing Decision Manager (SHDM). This loop happens in all the distributed instances of RIA at the same time. The interaction below shows how NFM publish the detected alert to the Newtwork IDS Events exchange. Then 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.

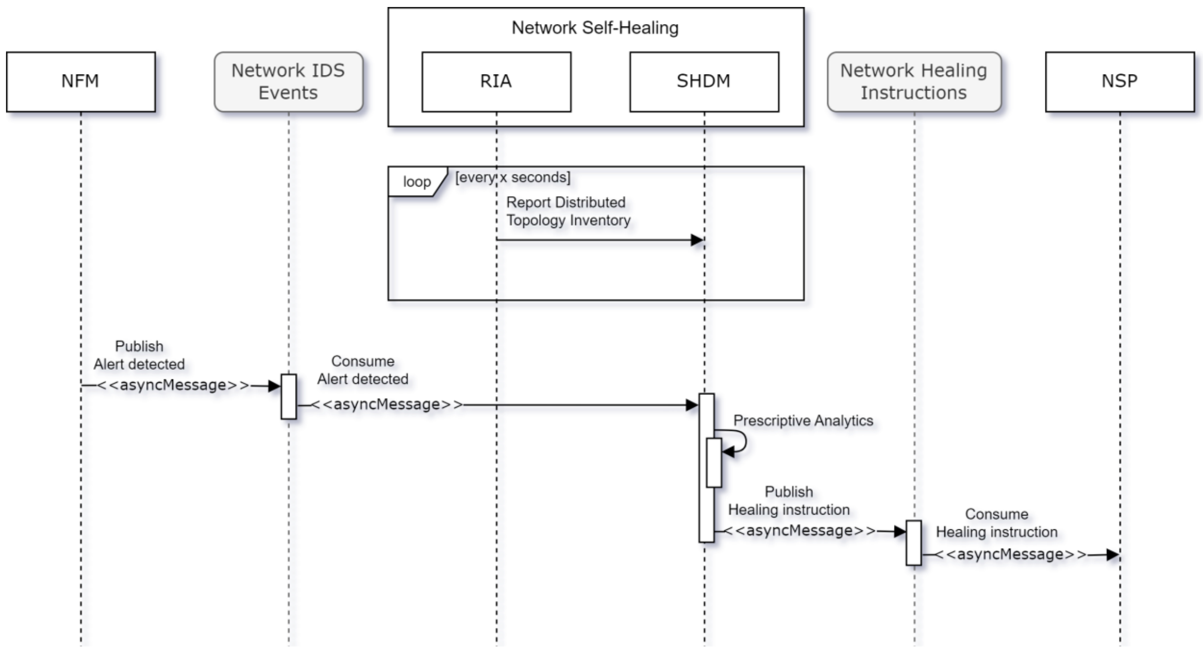

Network Self-Healing
NFM
Network IDS Events
RIA
SHDM
Network Healing Instructions
NSP
loop
[every x seconds]
Report Distributed Topology Inventory
Publish Alert detected <<asyncMessage>>
Consume Alert detected <<asyncMessage>>
Prescriptive Analytics
Publish Healing instruction <<asyncMessage>>
Consume Healing instruction <<asyncMessage>>

Figure 36 - 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	Definition	Channel	Exchange	Status
UWS	Network Flow Monitoring	RabbitMQ AMQP 0.9.1	Network IDS Events	Done
UWS	Network Self-Protection	RabbitMQ AMQP 0.9.1	Network Healing Instructions	Done
UWS	Network Self-Protection	RabbitMQ AMQP 0.9.1	Network Self-Protection Confirmations	Done

3.6.3.4 Technical solution

3.6.3.4.1 Deployment architecture view

Figure 37 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 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.

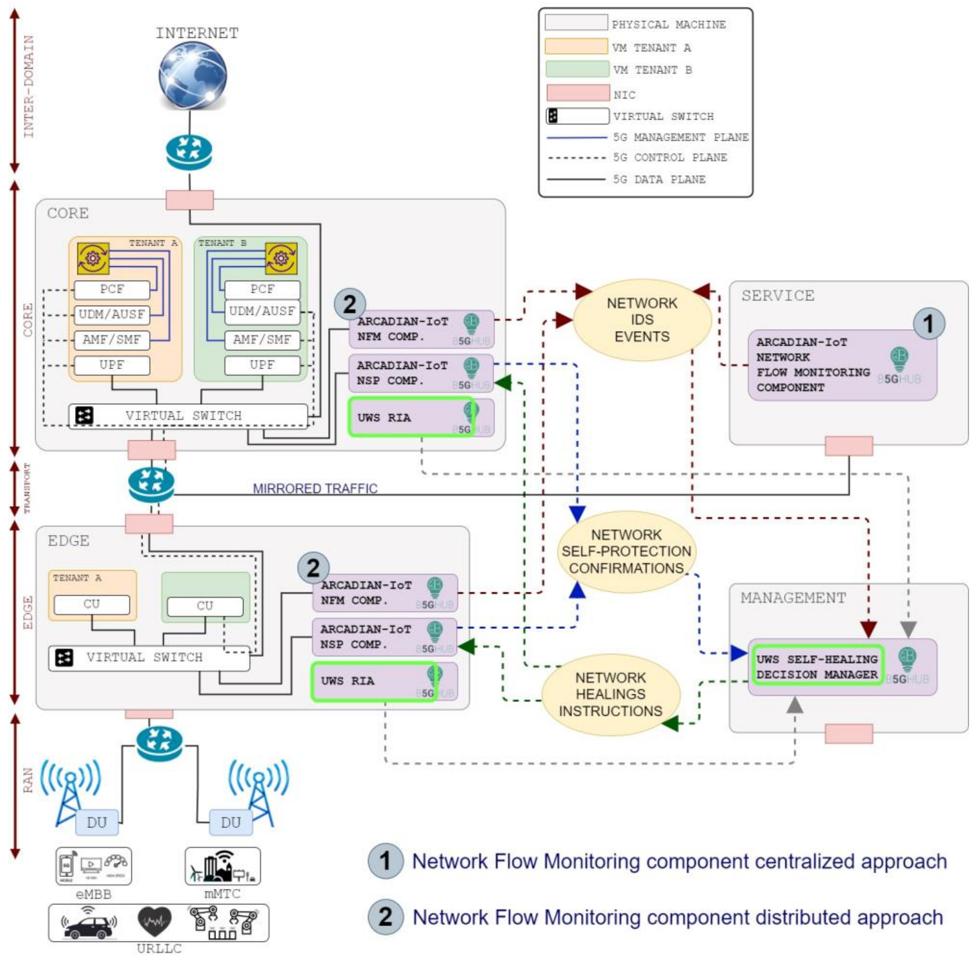


Figure 37 - Deployment architecture view of the Network Self-healing component

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

Following the prototyping, the overall behaviour of the Network Self-healing component will be empirically tested as a preliminary step towards the integration with the rest of the architectural components composing the Horizontal Planes of the ARCADIAN-IoT framework. The tests will be carried out in the cloud data centre at the UWS premises in Paisley Campus (Scotland) and will be focused on two aspects: functional validation and overall performance.

Regarding functionality, it will be empirically verified that the Network Self-healing component is able to effectively receive the alerts raised from the different Network Flow Monitoring component instances, that the RIA subcomponent is able to discover the topology, that the SHDM subcomponent is able to produce Prescriptive Analytics and that the Network Self-protection enforce in the data plane the set of healing actions sent by the Network Self-healing component able to communicate the Network Healing Instructions correctly with the final objective of stopping malicious traffic coming from DDoS attacks previously detected by the Network Flow Monitoring component.

Regarding performance, the following features will be evaluated to demonstrate the suitability of the component for its integration into a 5G IoT architecture such as the one defined in ARCADIAN-IoT:

- Scalability: Maximum Number of devices in the network infrastructure to be discovered by RIA.
- Delay: Average time to process each received alert and produce the network healing instruction.

3.6.5 Future work

The UWS team is currently working on the integration between the three components that belong to the cognitive self-protection network security loop. The development of the first version of the NSH component prototype is achieved and is being integrated with the rest of the ARCADIAN-IoT framework's components

After prototyping and as a last step in the development of the component, the prototype will be empirically validated and evaluated (as described in previous subsection). Hence, an empirical validation and evaluation of the overall performance of this component will be reported in deliverable D3.3. Finally, this component will be integrated into the ARCADIAN-IoT architecture where it will interact with other components to fulfil the project requirements. In this regard, it is particularly relevant the integration with the Network Flow Monitoring and the Network Self-protection components in order to achieve an autonomous cognitive self-healing loop in the network.

4 COMMON PLANE

4.1 Hardened Encryption (XLAB, BOX2M)

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 to 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. Furthermore, many devices in IoT setting are not capable of encrypting their own data, hence additional effort needs to be done for them to secure the data.

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 by one of the three hardware-based extensions (depending on the scenario): (i) the hardware-based Root of Trust (RoT) provided by an eSIM component, (ii) IoT devices with an embedded crypto chip, or (iii) independent / external crypto chip module integrated by vendors into its existing IoT device, as an add-on module. In order to support the 3 hardening approaches, 2 logical systems differing with respect to the hardware approach are being researched and implemented, which for the sake of simplicity we refer to as:

- Hardened Encryption with eSIM as RoT – research work led by XLAB
- Hardened Encryption with cryptochip system – led by BOX2M

The HE with eSIM 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 will include a decentralized key management system for distributing the (ABE) keys and access rights among the entities in the system. This will eliminate a single point of failure, such as a single server having access to all the cryptographic material. The key management system will be integrated with blockchain solution provided in ARCADIAN-IoT framework, and also with the framework’s authentication based on Self-Sovereign Identity and hardware-based RoT.

Thirdly, to increase the trust in the system, prevent impersonation attacks and similar breaches, the signing of encrypted payloads will be enabled by eSIM 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.

As for the Hardened Encryption with cryptochip system solution, tailored for industrial scenarios where IoT devices have limited or restricted capabilities, a specially designed system will be provided. The system, on one hand, includes crypto chips (as add-on module or embedded into an IoT device) enabling that the IoT device's firmware is equipped with (i) an agent for handling the keys and (ii) an agent for handling the decentralized authorization (as a second protection, on top of hardware encryption). On the other hand, the system also includes the IoT middleware platform (specific for the selected scenario) needed to include features for (i) key management (of crypto chip keys), (ii) ARCADIAN-IoT services management, and (iii) relaying on 3PP platforms and decentralised authorisation.

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).

KPIs with respect to the Hardened Encryption library with RoT Signatures:

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 (M30)	***Current value (M20)***
4 programming languages, 2 types of devices/platforms.	Currently supported Python, Java and Go, tested on an Android phone and Raspberry Pi.

KPI scope	
Add Root of Trust information to encrypted data with eSIM based signatures.	
Measurable Indicator	
1. Use of eSIM as RoT in hardened encryption processes (Y/N) 2. eSIM time to sign a payload (SHA256) 3. Number of different devices where the innovation is demonstrated	
Target value (M30)	***Current value (M20)***
1. Y	1. Y

2. RoT signature time: < 2 seconds (initial target; to be adjusted according to further research) 3. 2	2. RoT signature time: ~2.9 seconds 3. 1

KPI scope	
Provide secure and scalable key management and delegation synchronized with the decentralized identity management, multi-factor authentication and self-recovery.	
Measurable Indicator	
NA	
Benchmarking	
NA	
Target value (M30)	***Current value (M20)***
Decentralized key management integrated with other ARCADIAN-IoT components.	Decentralized key management not integration with other components.

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 (M30)	***Current value (M20)***
Comparable to the state-of-the-art benchmarks on all the devices.	On a laptop with 1.70GHz AMD Ryzen 7 PRO 4750U with Go based implementation and Python binding: • Encryption 14ms for a policy with 5 attributes • Decryption 10ms for a policy with 5 attributes On a Raspberry Pi 3 Model B+ with 1.4GHz Cortex-A53 with Go based implementation and Python binding: • Encryption 4.7s for a policy with 5 attributes

	• Decryption 3.1s for a policy with 5 attributes

KPIs with respect to the Hardened Encryption with cryptochip system:

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 (M30)	***Current value (M20)***
Less than 5min provisioning (for any scenario) into device side, with GUI in place and how to procedure, into device side. Less than 2min provisioning (for any scenario) into device side, with web dedicated page in place and how to procedure, into middleware side.	Less than 15min provisioning (for any scenario) into device side. Less than 5min provisioning (for any scenario) into middleware side.

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. This indicator is applicable for sense device – to – IoT platform.	
Target value (M30)	***Current value (M20)***
T1_E lower than 2sec T4_D lower than 3sec T3_E lower than 5sec T2_D lower than 4sec	T1_E lower than 3sec T4_D lower than 4sec T3_E lower than 6sec T2_D lower than 5sec

4.1.2 Technology research

4.1.2.1 Background

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.

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.

In the recent years, various project 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.

eSIM as Root of Trust (RoT)

The main background for the use of eSIM as hardened encryption process is GSMA IoT SAFE. This state-of-the-art GSMA working group and specification are still young and open for contributions. It 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 allowing 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 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.

Cryptochip 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 and Self-recovery,. 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 (P1) industrial IoT device, a crypto chip designed by STMicroelectronics was chosen, which is by default an authority for issuing Root of Trust products.

4.1.2.2 Research findings and achievements

We separate the research findings into two subsections according to the two subcomponents of the HE component. The first describes the research with respect to the *encryption library based on the ABE paradigm enhanced with the eSIM signatures*. The second is a c*rypto chip system* aiming at computationally limited devices, that are not able to support cryptographically more complex operations. The two subcomponents function *independently*, since they aim at different IoT settings. In particular, the first subcomponent will be evaluated in WP5 in Domains A and C involving devices such as drones, phones, and medical IoT devices, while the crypto chip system will be evaluated in Doman B.

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 will be 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 eSIM to anchor the RoT in hardware, these two technologies were chosen to harden the standard approaches in IoT for encrypting and authenticating the data.

In the joint studies made by TRU and XLAB, several hypotheses/approaches were considered on how to join ABE and eSIM. Examples are: (i) having the encryption mechanisms within the secure element itself; or (ii) having it establishing a secure communication channel for IoT devices. Not considering ARCADIAN-IoT context, both hypotheses were feasible: the eSIM has cryptographic 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 intended HE process that is described as an encryption process with RoT information (grant agreement). Considering the context and the objectives, the hypothesis selected for being prototyped for assessment and further research, is a novel combination of the eSIM abilities with the ABE previously described and summarized in Figure 38. It consists of using eSIM 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 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).

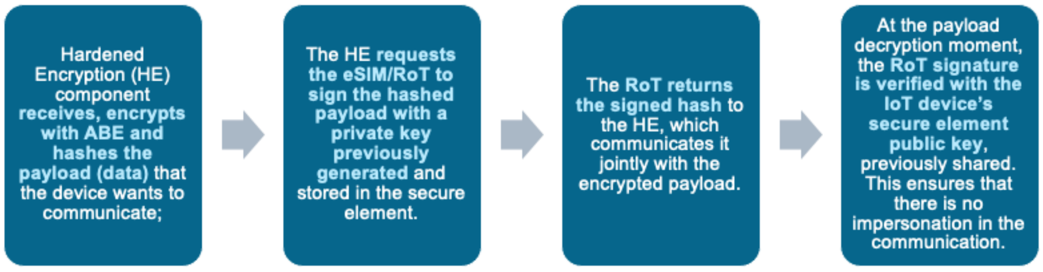

Hardened Encryption (HE) component receives, encrypts with ABE and hashes the payload (data) that the device wants to communicate;
The HE requests the eSIM/RoT to sign the hashed payload with a private key previously generated and stored in the secure element.
The RoT returns the signed hash to the HE, 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 38 - Research hypothesis for the Hardened Encryption to join ABE with eSIM as RoT

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

- Software implementation of ABE functionalities
- eSIM applet running in a secure element of a device enabling RoT signatures
- A middleware allowing to invoke the applet from the software part

The key generation in the hardware secure element can happen invoking the middleware aforementioned, or via secure GSM Over-The-Air 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. We describe the research findings and high-level description here, while we give technical specification in Section 4.1.3.

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 use, 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. First tests are in favour of this goal. Due to 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 39. In fact, this allows as to use ABE a variety of scenarios, such as encrypting text, figures, streams of data, etc., with minimum computational expense.

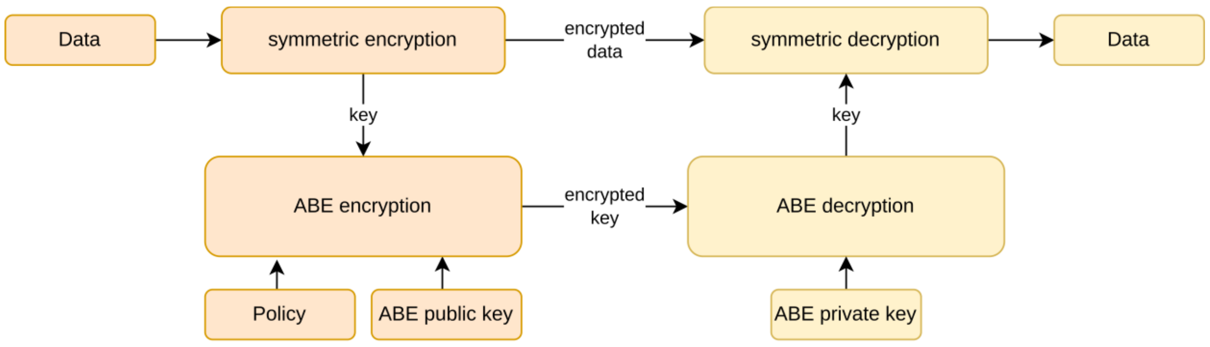


Figure 39 - The encryption and decryption process in HE

eSIM as hardware-based RoT for Hardened Encryption

A novel eSIM profile, ARCADIAN-IoT eSIM profile with a new security applet based on IoT SAFE specifications and architecture has been designed and is being prototyped for the Hardened Encryption component, as well as for Attestation and to receive security information and perform self-protection and self-recovery. This artifact, whose general architecture can be seen Figure 40 (including the applet following the IoT SAFE specs, explained before) starts by being a regular eSIM profile for connectivity enablement, including the regular TRU's MVNO (Mobile Virtual Network Operator) connectivity features for such, like the eUICC OS or SIM OS, the NAA (Network Access Applications), and a set of applets that are able to run specific processes in the secure element, some of which specific from TRU (e.g. its multi-IMSI patented technology for programmatic management of worldwide connectivity attachment). Despite GSMA's IoT SAFE architecture focuses eSIM (with the mentions to eUICC), we don't forsee any reason for the new technology developed to not work in any SIM factor (SIM, eSIM, iSIM, …). However, the physical characteristics of the embedded or integrated forms (eSIM and iSIM) are, by design, more secure, because it is much harder to use a specific secure element (with specific 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).

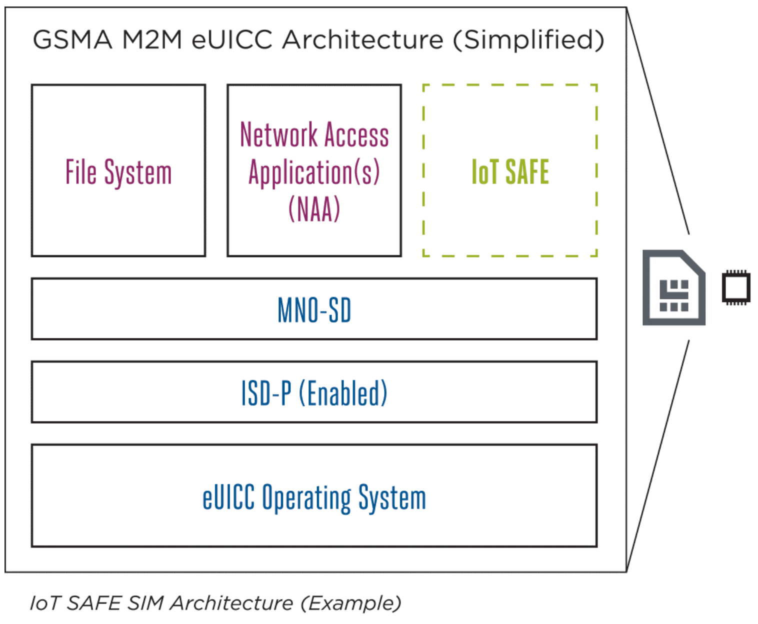

GSMA M2M eUICC Architecture (Simplified)
File System
Network Access Application(s) (NAA)
IoT SAFE
MNO-SD
ISD-P (Enabled)
eUICC Operating System
IoT SAFE SIM Architecture (Example)

Figure 40 eUICC structure comprising the IoT SAFE [17]

As previously mentioned, the novelty of ARCADIAN-IoT eSIM security applet starts by its compliance with GSMA IoT SAFE[16] (SIM Applet For Secure End-to-End Communication), which may ensure compliance with industry standards (at least for its baseline developments). To these standards, are added the developments needed for the integration with ARCADIAN-IoT Hardened Encryption process, to novel remote attestation procedures, device self-protection and device self-recovery processes, none of which is a known scenario in the state of the art for the use of the SIM as RoT. TRU research team is looking forward to contribute to the enhancement of the existent state of the art, being already present in GSMA IoT SAFE working group to contribute with the outcomes of the research. The intended scenarios for the eSIM ARCADIAN-IoT profile in a real-word deployment would be the following:

- *M2M bootstrap scenario*: When a compliant IoT device is turned on for the first time it is provisioned with an ARCADIAN-IoT eSIM profile, which includes the connectivity features and the security applet. Once the profile is activated in the device it connects to TRU network, which, by recognizing the device and the ARCADIAN-IoT eSIM, 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.
- *Personal device bootstrap scenario*: When a user registers in an ARCADIAN-IoT compliant app it is provisioned with an ARCADIAN-IoT eSIM profile, which includes the connectivity features and the security applet. Once the profile is activated in the device it connects to TRU network, which, by recognizing the device and the ARCADIAN-IoT eSIM, 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.
- *Hardened Encryption process*: To ensure data privacy and security, all the sensitive outgoing data from a compliant IoT device, or compliant 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 (or its hash), and finally sends it to the intended service.

- *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 are a vision that can be enhanced throughout project, but already guide the research.

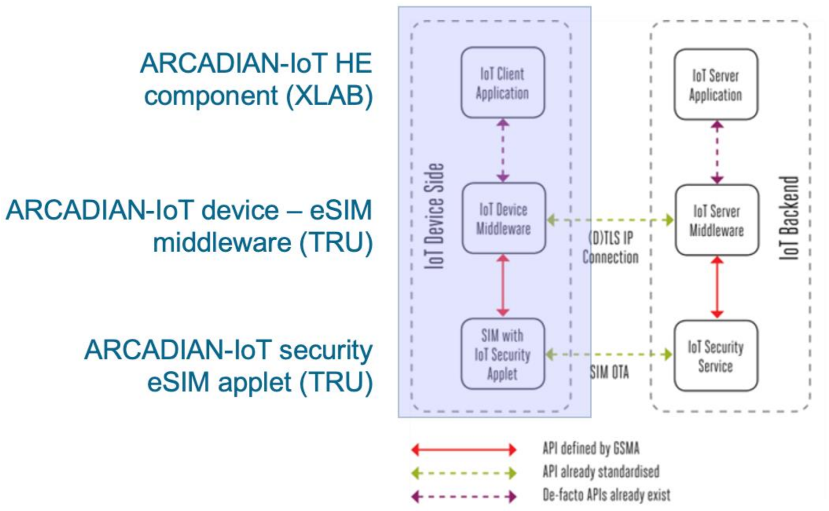


Figure 41 - ARCADIAN-IoT Hardened Encryption prototype mapped into GSMA IoT SAFE architecture[26]

Figure 41 depicts the match between ARCADIAN-IoT components with GSMA IoT SAFE components. In the current prototype, we are focusing on the *IoT Security Applet*, on the *IoT Device Middleware*, which allows the communication with the applet, and on the integration with the *IoT Client Application*. The IoT Client Application in this case, is the Hardened Encryption component. Specifically, the first prototype of the RoT comprises the following methods, following GSMA IoT SAFE specifications.

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

#	Method	Brief description	Prototyping status
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 key store with their respective value and the public key is returned to the caller.	First prototype done for the applet and middleware. Middleware available in Python, considering the needs of Domain A, in their drone setup. Successfully tested in two different modems (the communication between the middleware and the applet is made through the modem).
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.	First prototype done for the applet and middleware. Integrated testing within the Hardened Encryption process successful.
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, in this case, get the signed hash from the SIM applet (to be used after 3)	

Table 14 - Methods from the first prototype of ARCADIAN-IoT security eSIM applet made accessible by the existent device middleware

The methods in Table 14 are the key ones for fulfilling the intended functionality of the RoT in the HE process. These were already prototyped, and target of successful unit and functional testing, and integration testing within the HE process. Some results are presented in section 4.1.4.

Decentralized key management of ABE keys and eSIM 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.
- eSIM public keys: the signature verification keys of the devices participating in the platform, corresponding to private signing keys embedded in eSIM.

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 an architecture plan that enables to decentralize the trust in the cryptosystem, by introducing multiple improvements:

- We use CP-ABE encryption scheme that uses a multiple key authority setup (due to A. Lewko and B. Waters [21]) 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 42 below. Authorities 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 decentralized ABE schemes in the context of IoT devices.

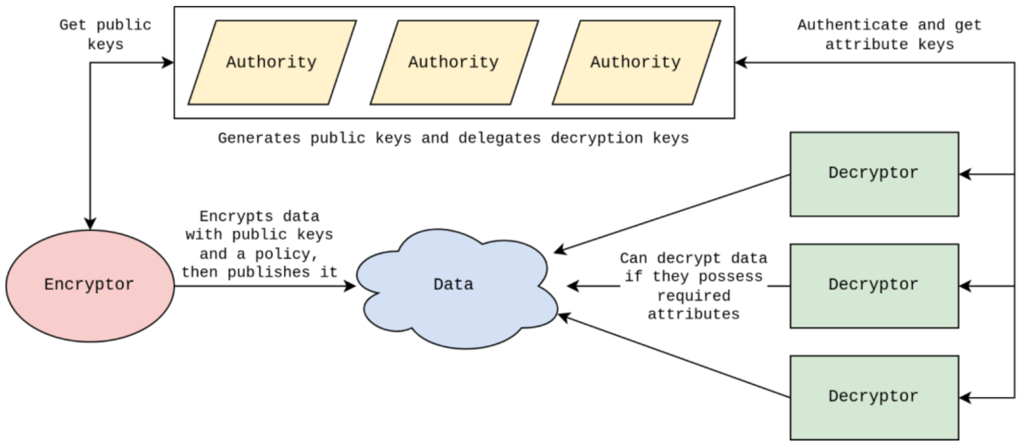

Get public keys
Authority
Authority
Authority
Authenticate and get attribute keys
Generates public keys and delegates decryption keys
Decryptor
Encrypts data with public keys and a policy, then publishes it
Encryptor
Data
Can decrypt data if they possess required attributes
Decryptor
Decryptor

Figure 42 - Multi-authority key management

- We include in the architecture the integration of the HE component with ARCADIAN-IoT blockchain ledger for publishing public parameters 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, see Figure 42. In particular, the public ABE keys will be put on the ledger, allowing the ARCADIAN-IoT entities to encrypt data for another receiving device or group. The signing public keys needed to verify the authenticity of encrypted data will be put on the ledger as well.
- We include 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.
- 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.

4.1.2.2.2 *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.

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 commands (1 other 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 is pending the delivery from factory), and 6 were tested completely (GSM, UMTS, LTE, WiFi, ETH, and ETH & WiFi combined one); all mobile broadband boards have by default an eSIM too, so tests were done with both a regular SIM and an eSIM; 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 eSIM and regular SIM, and to manage balanced the power supply management and internal communication channels between motherboards and telecommunication systems

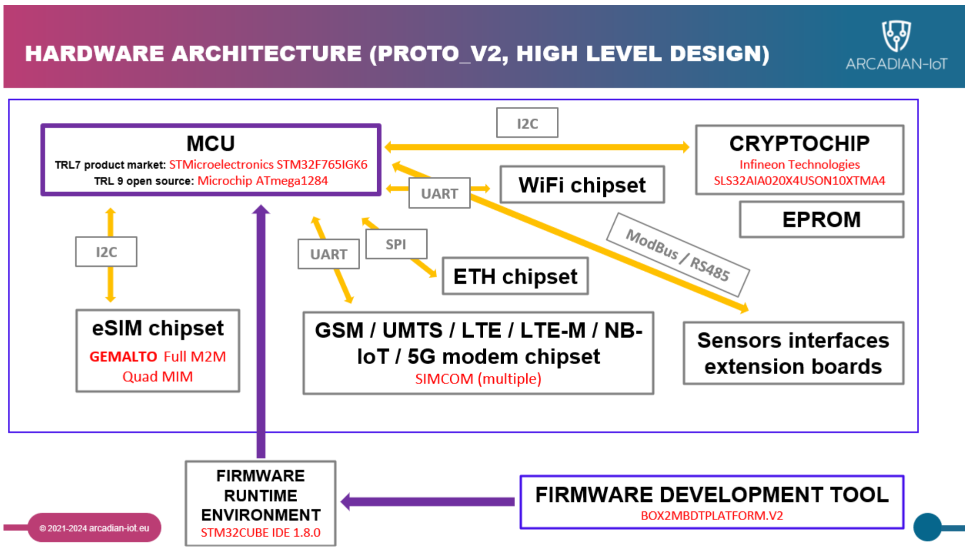

HARDWARE ARCHITECTURE (PROTO_V2, HIGH LEVEL DESIGN)
ARCADIAN-IoT
MCU
TRL7 product market: STMicroelectronics STM32F765IGK6
TRL 9 open source: Microchip ATmega1284
I2C
CRYPTOCHIP
Infineon Technologies
SLS32AIA020X4USON10XTMA4
UART
WiFi chipset
EPROM
I2C
UART
SPI
ETH chipset
ModBus / RS485
eSIM chipset
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 43 - Hardware high level architecture

Firmware challenges and achievements

The main challenge was to deploy a whole 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.

Middleware challenges and achievements

The main challenges were to deploy the subcomponent as a modular component, Docker contained, API ready with any 3PP system, with API configuration options to make the integration with Remote Attestation, Reputation, Self-aware Data Privacy or Network Flow Monitoring systems. 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".

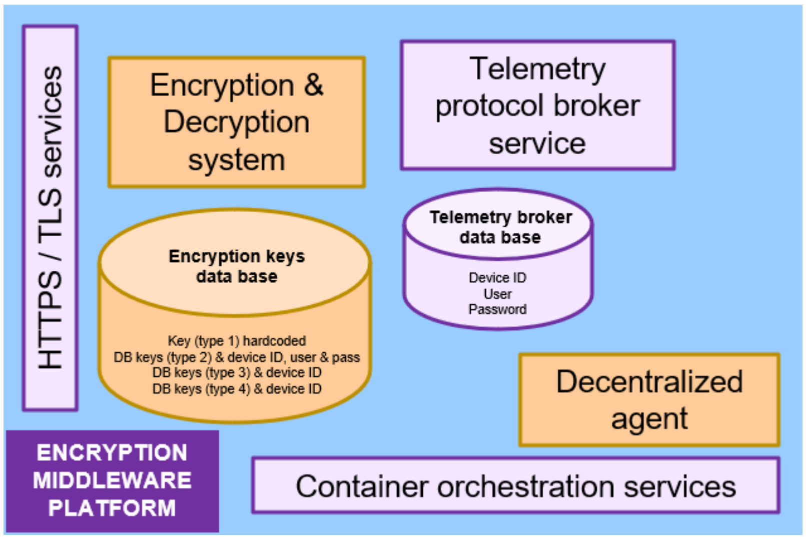

HTTPS / TLS services
Encryption & Decryption system
Telemetry protocol broker service
Encryption keys data base
Key (type 1) hardcoded
DB keys (type 2) & device ID, user & pass
DB keys (type 3) & device ID
DB keys (type 4) & device ID
Telemetry broker data base
Device ID
User
Password
Decentralized agent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

Figure 44 - Middleware architecture

4.1.2.3 Produced resources

In this section we report on the current implementation status of the HE component and the functionalities of the first complete prototype.

4.1.2.3.1 Hardened Encryption with RoT Signatures

With respect to the HE library with ABE encryption and eSIM signatures, the main development effort was put into implementing the first prototype. In particular, the main goal for the reporting period was to demonstrate the use of it using a simple Raspberry Pi device with a SIM. The development of the key management subcomponent is currently limited to providing basic functionalities needed for the use of the HE library and will be developed further in the next reporting period, mostly due to its interconnectedness with other ARCADIAN-IoT components. The goal was overachieved:

- We have implemented a Go based (ABE) encryption library, together with Python and Java bindings. The library was developed with modern paradigm of continuous testing, tools for automatic detection of security issues, and was tested on multiple devices including servers, phones and a Raspberry Pi. It is available to partners at https://gitlab.com/arcadian_iot/hardened_encryption. We provide the documentation as well as explanations and examples on how to install and use the library. We detail the API specifications of it in Section 4.1.3.
- The SIM security applet has also been developed and its implementation is independent of the device where the SIM will live (applicable to any SIM factor). It currently has all the envisioned needed functionalities, but may need to have performance optimizations (e.g. time to sign the hash of the encrypted payload). The test results and its analysis can be found in section 4.1.4.
- The device middleware, for the HE communication with the SIM secure element also has a first prototype, developed considering the IoT device (drones) in Domain A. To allow for the integration in that use case, it is currently developed in Python. For the moment and until the exploitation strategy is fully clear, nor the applet or the middleware will be released as open source. It will just be provided as needed for ARCADIAN-IoT integration and demonstration purposes. Both the SIM security applet and the device middleware were successfully tested, integrated in the HE setup, with 2 different modems (which may influence the middleware development (AT) commands or the applet behaviour).
- Basic key management authorities have been implemented in Go and can be deployed directly or as Docker containers. They currently provide basic functionalities with respect to the distribution of ABE private keys, but have not been integrated with the other ARCADIAN-IoT components
- The use of subcomponents has been jointly demonstrated in a demo, where a Raspberry Pi device obtains all the crypto material from the key management authorities, encrypts data with an ABE policy, signs it in its eSIM security applet, and forwards the data to a data recovery server. A device with appropriate attributes is then able to recover and decrypt the data. The source code is provided at https://gitlab.com/arcadian_iot/hardened_encryption/-/tree/main/ahe-demo.

4.1.2.3.2 Hardened encryption with crypto chip system

With respect to the crypto chip-enabled hardened encryption, the highlighted produced resources are:

- Industrial IoT hardware device (see Figure 45)
- Device firmware
- Encryption & decryption firmware agent
- Middleware software application
- Encryption & decryption middleware agent
- API configurator
- API integration with a live IoT platform, to have the test bed operational end-to-end
- A complete IoT stress test environment (hardware set-up, cloud set-up, service operation center, defined measurements)

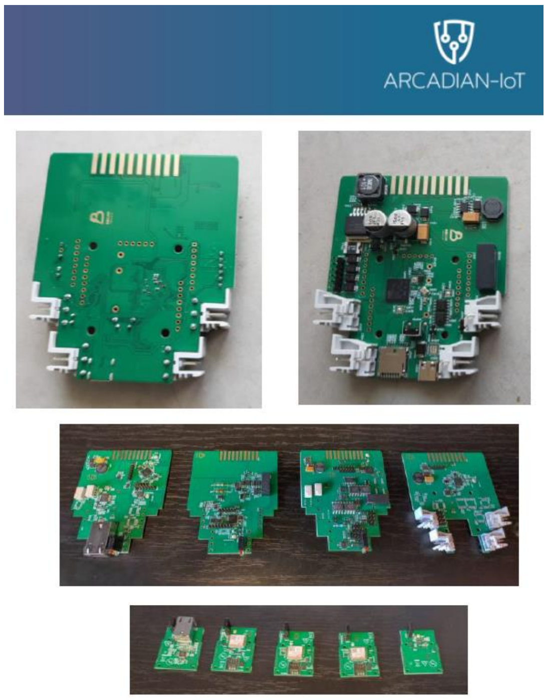


Figure 45 - Hardware prototypes

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

4.1.3.1.1 Hardened Encryption with RoT Signatures

Architecture view of the HE with ABE encryption and eSIM signatures is visualized in Figure 46.

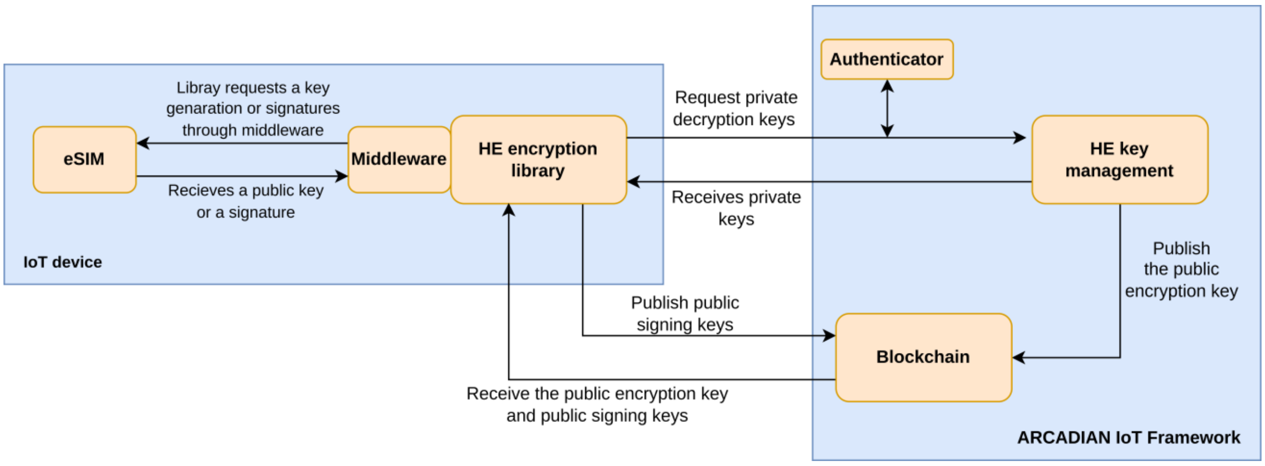

Libray requests a key genaration or signatures through middleware
eSIM
Middleware
HE encryption library
Recieves a public key or a signature
IoT device
Request private decryption keys
Receives private keys
Authenticator
HE key management
Publish the public encryption key
Publish public signing keys
Blockchain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

Figure 46 - Logical architecture view of HE

- HE library is a software library developed in Go language with binding to Python, Java (currently), C, JS (to be provided) 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 eSIM 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 eSIM subcomponent through the modem (via AT commands)
- ARCADIAN-IoT eSIM 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, whose secrets are stored in the hardware secure element, to cryptographically sign data with RoT.
- HE key management is a component handling key distribution. It will be deployed as multiple authorities running at decentralized servers and be generating private and public keys that the devices will need to use the HE library.

4.1.3.1.2 Hardened encryption with crypto chip system

The architecture of the Hardened Encryption with the crypto chip system is presented in Figure 47.

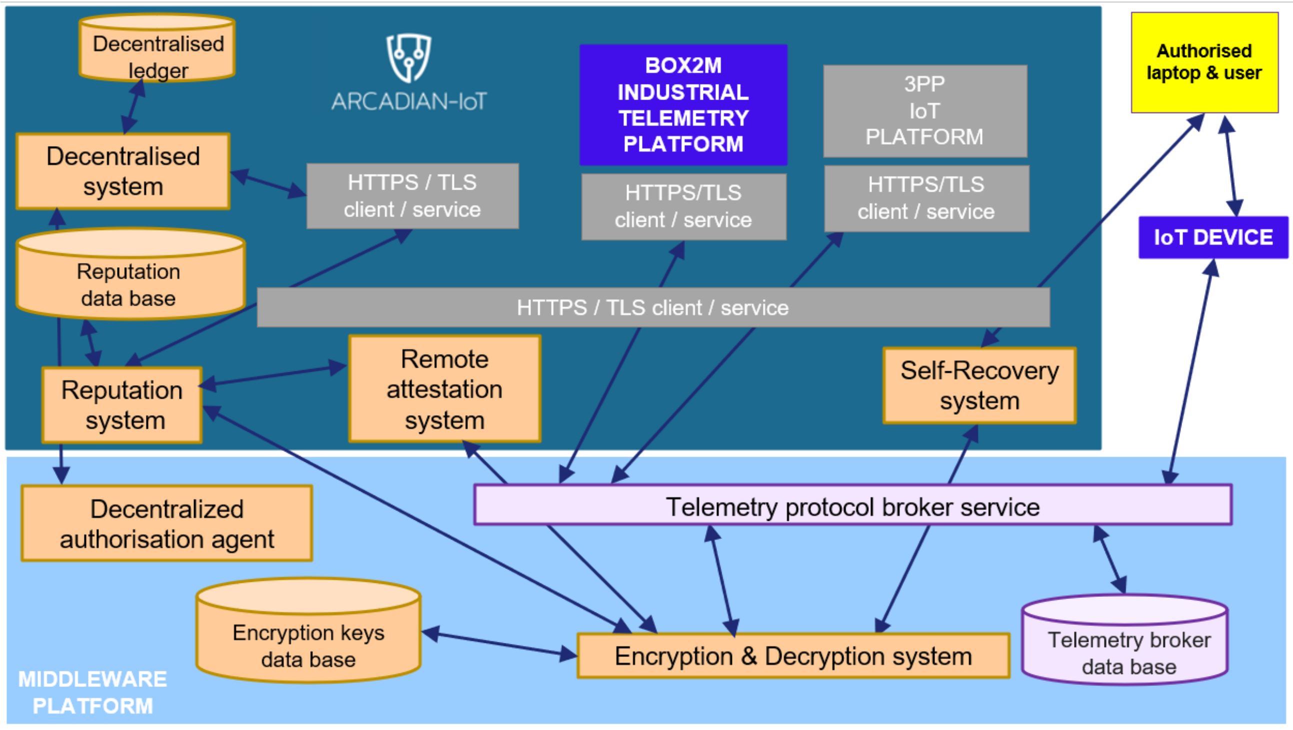

Decentralised ledger
ARCADIAN-IoT
BOX2M INDUSTRIAL TELEMETRY PLATFORM
3PP IoT PLATFORM
Authorised laptop & user
Decentralised system
HTTPS / TLS client / service
HTTPS/TLS client / service
HTTPS/TLS client / service
IoT DEVICE
Reputation data base
HTTPS / TLS client / service
Reputation system
Remote 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 47 – Hardened Encryption (with crypto chip) system architecture

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.

eSIM security applet and HE in device self-protection and device-self recover

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 this first prototype, 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 already 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 eSIM 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 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 D4.2). This component is to live in a 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

4.1.3.3.1 Hardened Encryption with RoT Signatures

The sequence diagram in Figure 48 explains how the HE subcomponents for the ABE encryption and eSIM 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 eSIM a public signing key, that is then 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 eSIM to obtain RoT signatures. In the decryption phase the encrypted data is first verified, where the public key is checked on the permissioned blockchain, and then decrypted.

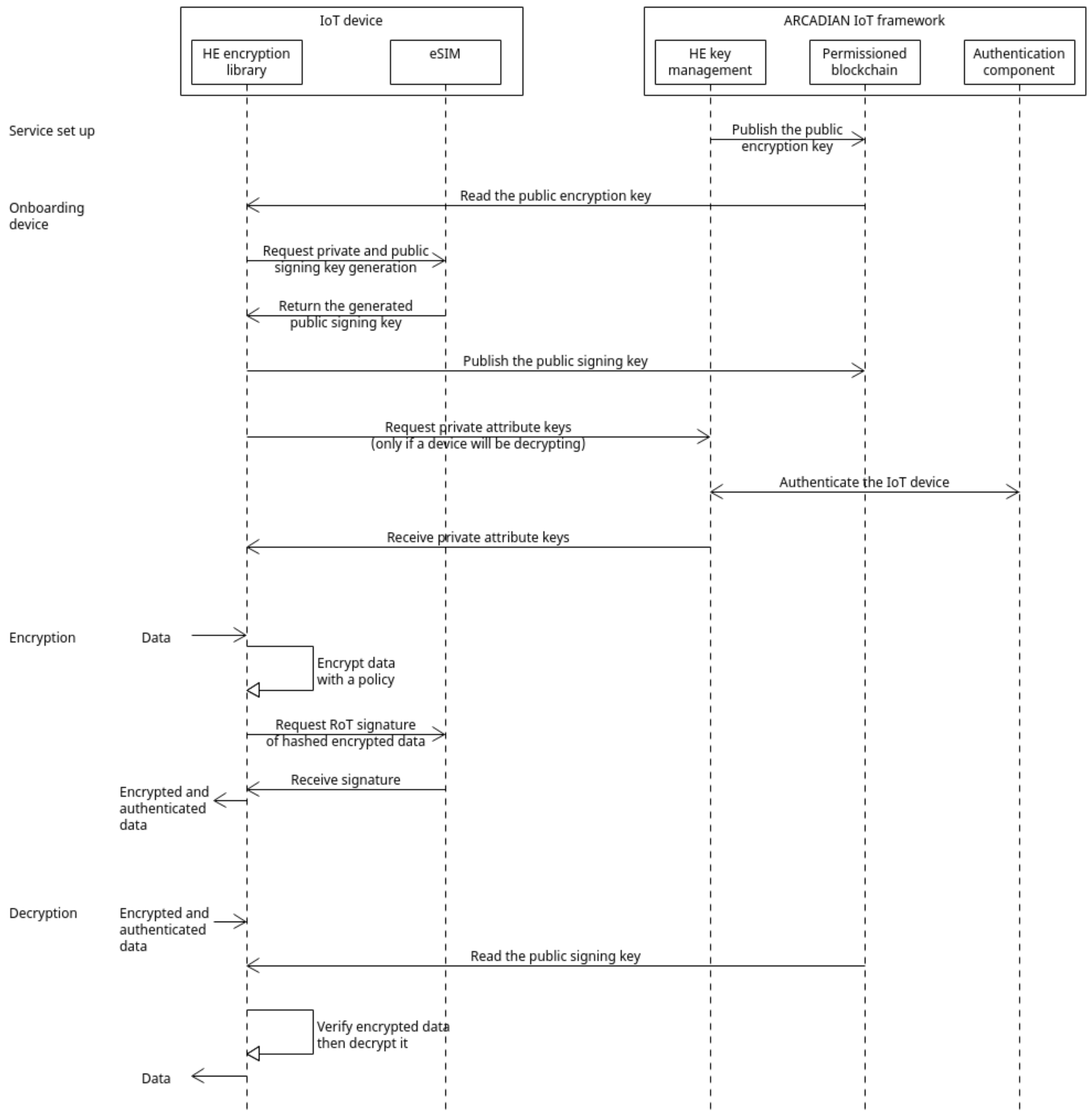

IoT device
HE encryption library
eSIM
ARCADIAN IoT framework
HE key management
Permissioned blockchain
Authentication component
Service set up
Publish the public encryption key
Read the public encryption key
Onboarding device
Request private and public signing key generation
Return the generated public signing key
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 48 - The sequence diagram of the HE component with internal and external interfaces

4.1.3.3.2 Hardened encryption with crypto chip system

The sequence diagram in Figure 49 and Figure 50 explains how the different subcomponents of the crypto chip-enabled Hardened Encryption interact during the onboarding and during traffic exchange.

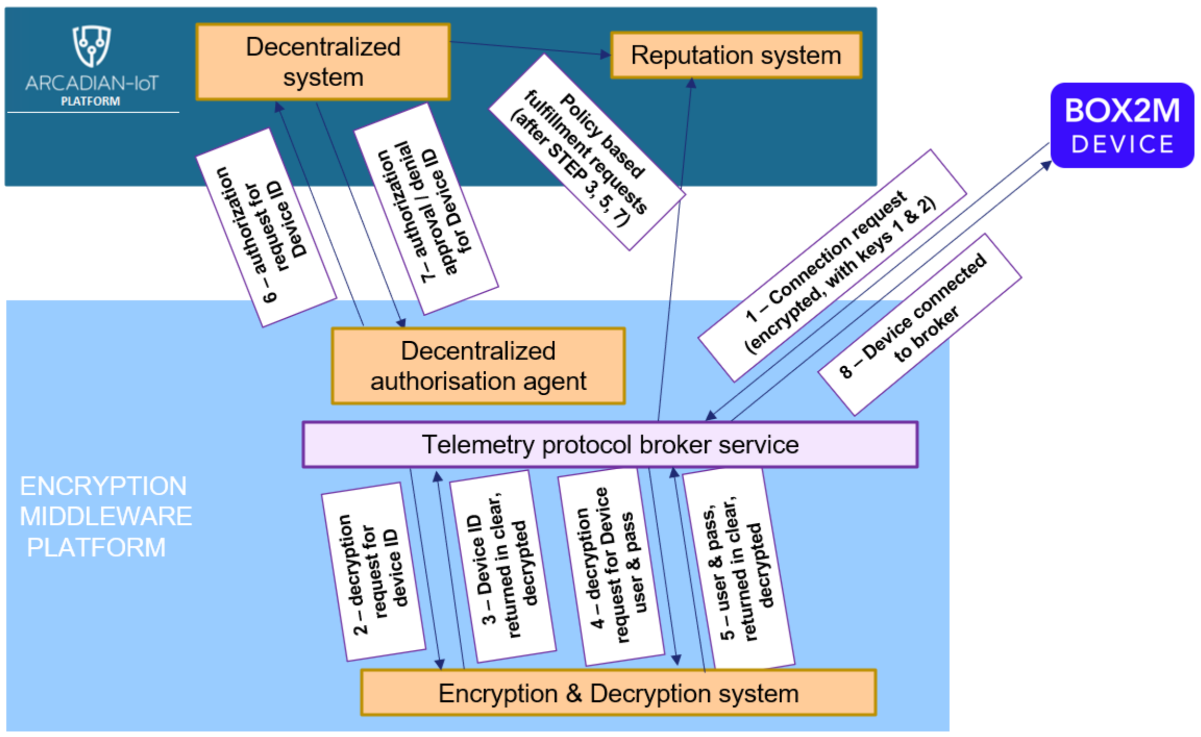

ARCADIAN-IoT PLATFORM
Decentralized system
Reputation system
BOX2M DEVICE
6 – authorization request for Device ID
7 – authorization approval / denial for Device ID
Policy based fulfillment requests (after STEP 3, 5, 7)
1 – Connection request (encrypted, with keys 1 & 2)
8 – Device connected to broker
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 49 - Device onboarding

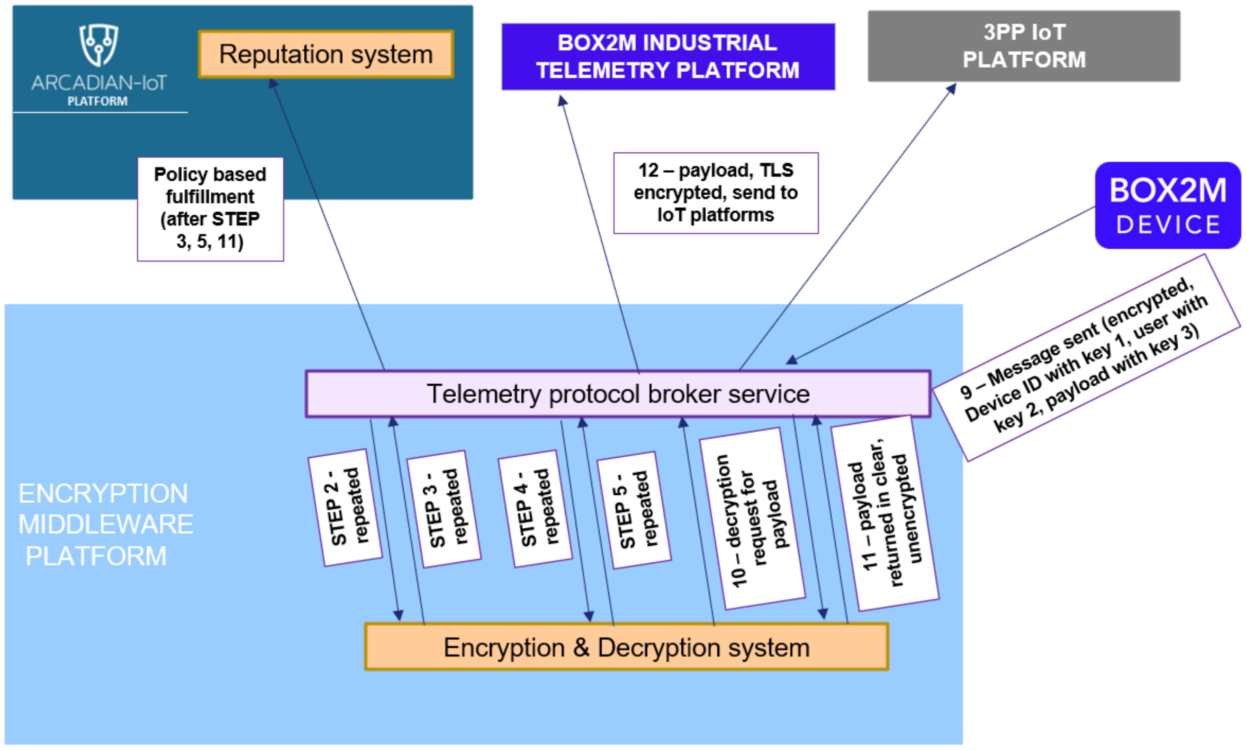

ARCADIAN-IoT PLATFORM
Reputation system
BOX2M INDUSTRIAL TELEMETRY PLATFORM
3PP IoT PLATFORM
Policy based fulfillment (after STEP 3, 5, 11)
12 – payload, TLS encrypted, send to IoT platforms
BOX2M DEVICE
9 – Message sent (encrypted, Device ID with key 1, user with key 2, payload with key 3)
Telemetry 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 50 - Device traffic management

4.1.3.4 Interface description for Hardened encryption with RoT signatures

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

- The 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.

- The 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.
- The HE component is used, besides by the devices in the framework, by two other ARCADIAN-IoT components:
    - Remote Attestation component requires the attesters to use the HE encryption library (and through it eSIM) 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.

4.1.3.5 Technical solution – Hardened Encryption with RoT signatures

4.1.3.5.1 Deployment architecture view

While the deployment view of the HE subcomponent for the ABE encryption and eSIM signatures has been shown in Figure 48, we report in Figure 51 how the crypto chip system deployment is designed in the Domain B.

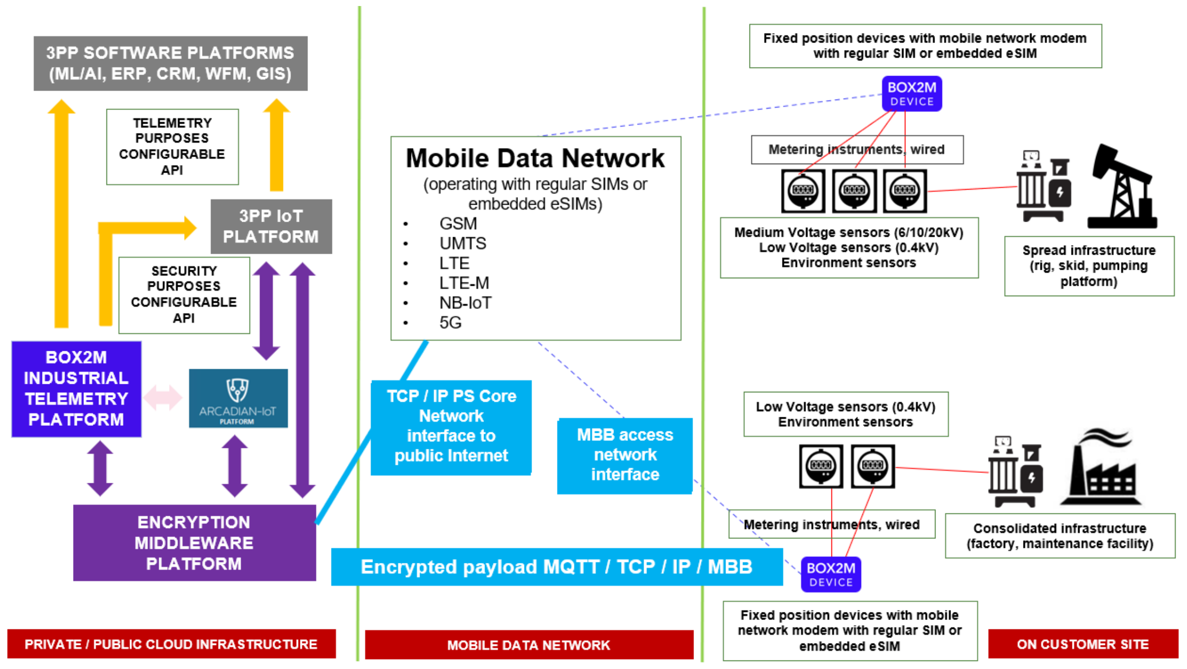

3PP SOFTWARE PLATFORMS (ML/AI, ERP, CRM, WFM, GIS)
TELEMETRY PURPOSES CONFIGURABLE API
3PP IoT PLATFORM
SECURITY PURPOSES CONFIGURABLE API
BOX2M INDUSTRIAL TELEMETRY PLATFORM
ARCADIAN-IoT PLATFORM
ENCRYPTION MIDDLEWARE PLATFORM
Mobile Data Network
(operating with regular SIMs or embedded eSIMs)
GSM
UMTS
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 51 - Grid domain implementation of crypto chip system

4.1.3.5.2 API specification for hardened

In this section we detail the interface to the functionalities of HE. In particular, we focus on the Python bindings, which have reached an almost stable version, and are expected to be validated in Domain A use case of emergency and vigilance service with drones. Information about the other interfaces (Java, C…) will be provided in the final report, while it is partially available online. The implementation with the documentation and installation steps are available to partners at https://gitlab.com/arcadian_iot/hardened_encryption. Here we outline the main points.

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. These services are provided by the eSIM security applet through the device middleware provided for that purpose. The security applet will be in a eSIM/SIM profile, which it is placed in the devices when the eSIM with it is provisioned or the SIM card with it is inserted in the device. The device middleware is envisioned to be compiled jointly with the remaining code from the HE component.

HE library

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 eSIM, 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 an eSIM or 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])
```

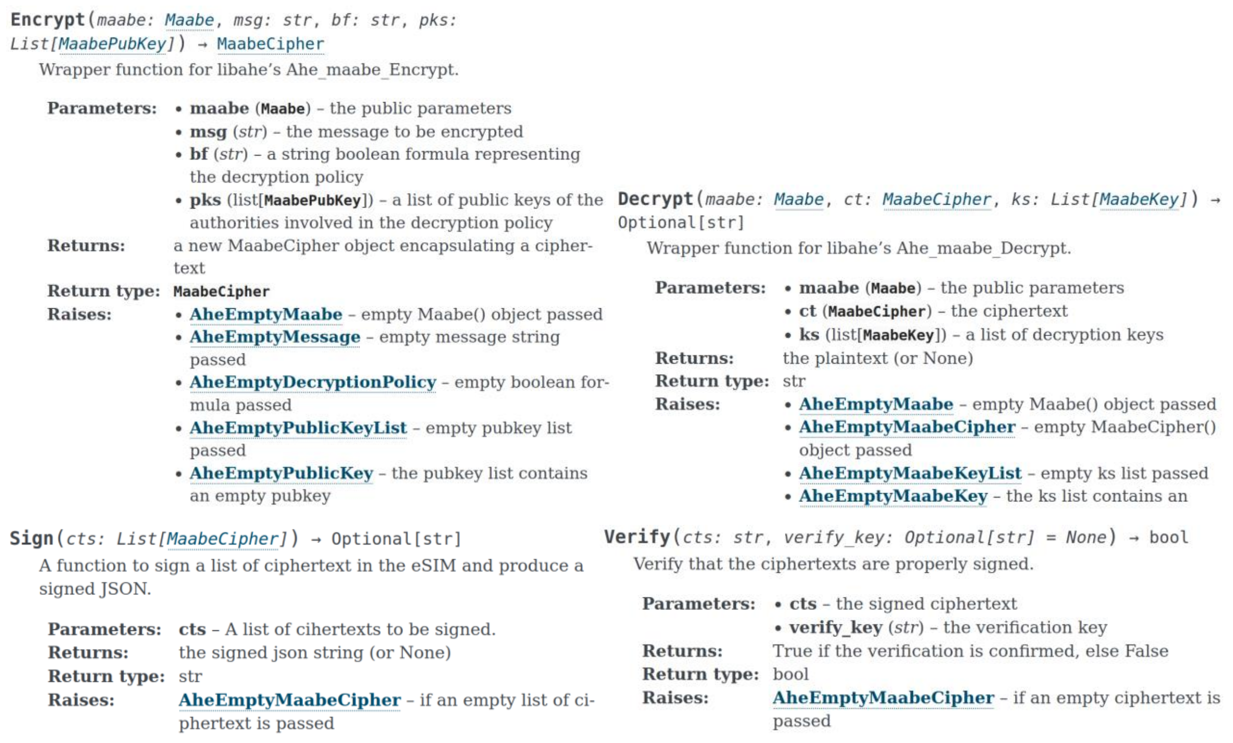
A receiving device can validate the authenticity of data (directly on the ciphertext)

```
# 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 52 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 52 - Snippets from the HE documentation

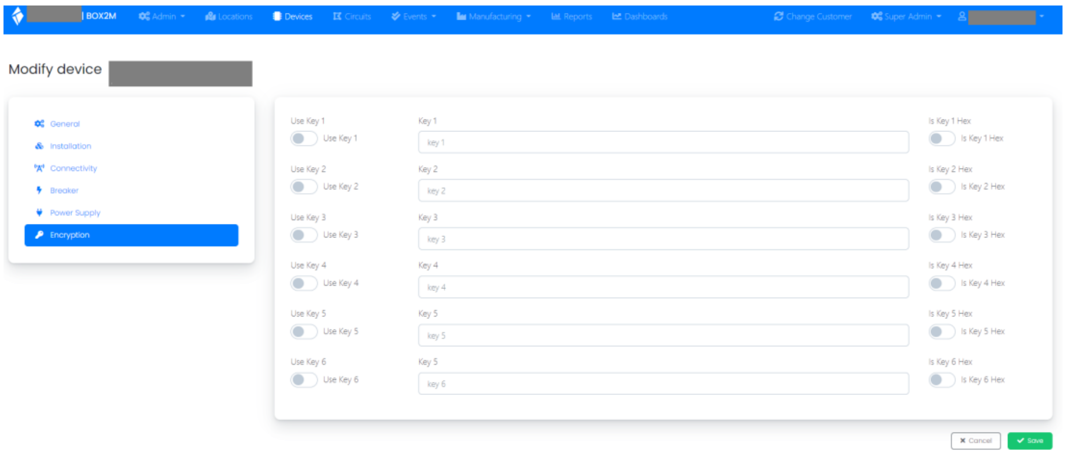
HE key management

We will provide a Dockerized version of this components. The specifications will be provided in the next reporting period. Currently, the component supports basic functionalities needed to support HE library.

4.1.3.6 Technical solution – Hardened Encryption with cryptochip system

4.1.3.6.1 Frontend design

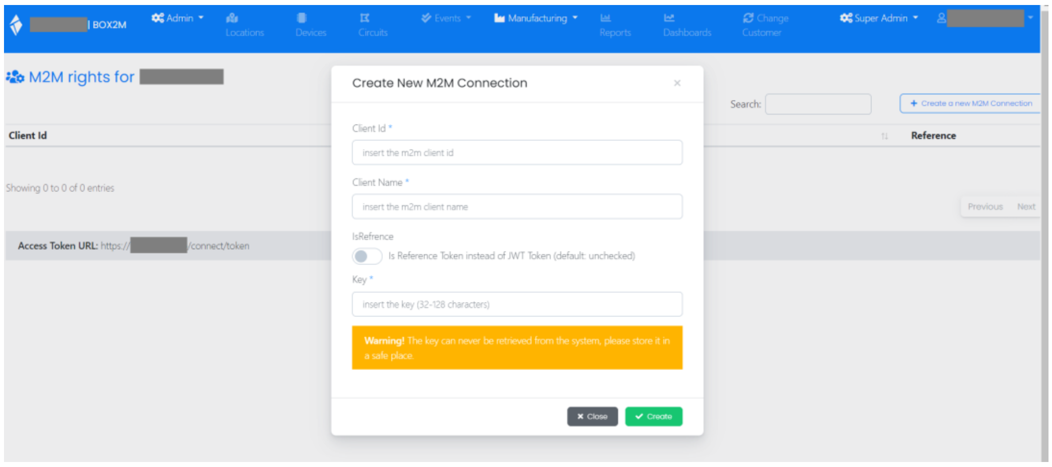
The crypto chip system includes a frontend for the key middleware and key-set up management, see Figure 53 and Figure 54.



Modify device [redacted]

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

Figure 53 - Crypto chip system – keys set-up (synced with device firmware provisioning)



M2M rights for [redacted]

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 54 - Crypto chip system - middleware API set-up

4.1.3.6.2 *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/eSIM 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).

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.

4.1.4 Evaluation and results

In this section we report on the main outcomes and results of the evaluation of the 2 Hardened Encryption approaches.
For the **Hardened Encryption supported by eSIM as RoT**, one of the main contributions provided in the reporting period is an ABE based library implemented in Go, cross compiled for multiple platforms, together with bindings (interfaces) in Python and Java. This is an important step in achieving the first KPI: "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."

One of the main reasons, to develop the encryption library starting with a Go implementation, is that the existent cryptographic libraries in Go provide a secure and performance-wise optimized basis for the implementation of high-level protocols. This helps us achieve the KPI: "Provide an efficient implementation of the process with encryption times comparable to the state-of-the-art results on multiple devices, and RoT signatures in acceptable time for the communication processes." 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 the current implementation and see 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, and on a Raspberry Pi 3 Model B+ with 1.4GHz Cortex-A53. Note that both 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 Table 15 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.

	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 (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 15 - Encryption and decryption overhead for using ABE

While a direct comparison to [22] cannot be made, since we are using a different ABE scheme and (currently) benchmarking it on a slightly more performant machine, the numbers are improved and indicating that the technology can be practical in IoT scenarios. Nevertheless, it seems that the performance on devices with limited capabilities could be improved, and we shall investigate it further if needed. To fulfil the mentioned KPI we need to benchmark the library on more devices, to prove the hypothesis that the library works in acceptable times on various devices.

As for the eSIM/SIM RoT signature process and the KPI “add Root of Trust information to encrypted data with eSIM based signatures”, it is known that the computing power of any SIM form is limited. In fact, the hypothesis formulation considered exactly this when TRU and XLAB decided that the SIM should sign just the hash (SHA256) of the encrypted payload and not the encrypted payload itself. In this prototype, no computational/algoritmic optimization was considered for the applet nor for the middleware, which is something that can be performed towards the final prototype. The performance of the current eSIM security applet to perform its role in the HE component is the following (average data from 10 executions):

A. Full flow, including starting and selecting the right SIM, selecting the right applet, the compute init command, the compute update command and the get response command took in average close to 6 seconds.
B. Just the compute signature and get response commands took an average close to 2.9 seconds.

As can be seen, for the moment we have disregarded the generate key pair command because it is not considered relevant for the normal operation of the HE component. Considering that the RoT is meant to be used to keep long-term secrets like the signature private key, the generate key pair is expected to be triggered in a quite low frequency (potentially just once), and can be the case that it is not triggered by the HE component (to be defined).

As for the measurement took, the A. flow allows to understand the time the applet would take to sign if the card was turned off at the beginning of the process. This can be the case in IoT devices that just communicate, e.g., once per day or once a week, and have limited battery. In this case, for devices that communicate with low frequency, around 6 seconds to sign a payload may be acceptable.

Regarding the B. flow, it can apply to the other cases, of devices that need to communicate sensitive data with high frequency (e.g., every 5 seconds). In this case, having the applet taking ~2.9 seconds to sign may be a short coming.

As previously mentioned, the ABE was integrated with the device middleware that allows the communication with the security applet. The functional tests were successful. This integration happened in a Raspberry Pi 3, Model B+, with 2 different modems, the SIMCom SIM7600G-H-M2 R2 (LTE Cat4 M2), and the Monarch GM01Q (LTE Cat M1). The reason to perform the tests with 2 modems is due to the fact that the communication between the middleware and the security applet happens through the modem, with AT commands. The performance tests to the SIM as RoT in the HE processes were performed in this setup.

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
- eSIM provisioning on mobile broad band boards and testing, based on the board pluged in and engaged
- Regular SIM (multiple operators, including a MVNO) provisioning on mobile broad band boards and testing, based on the board pluged 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

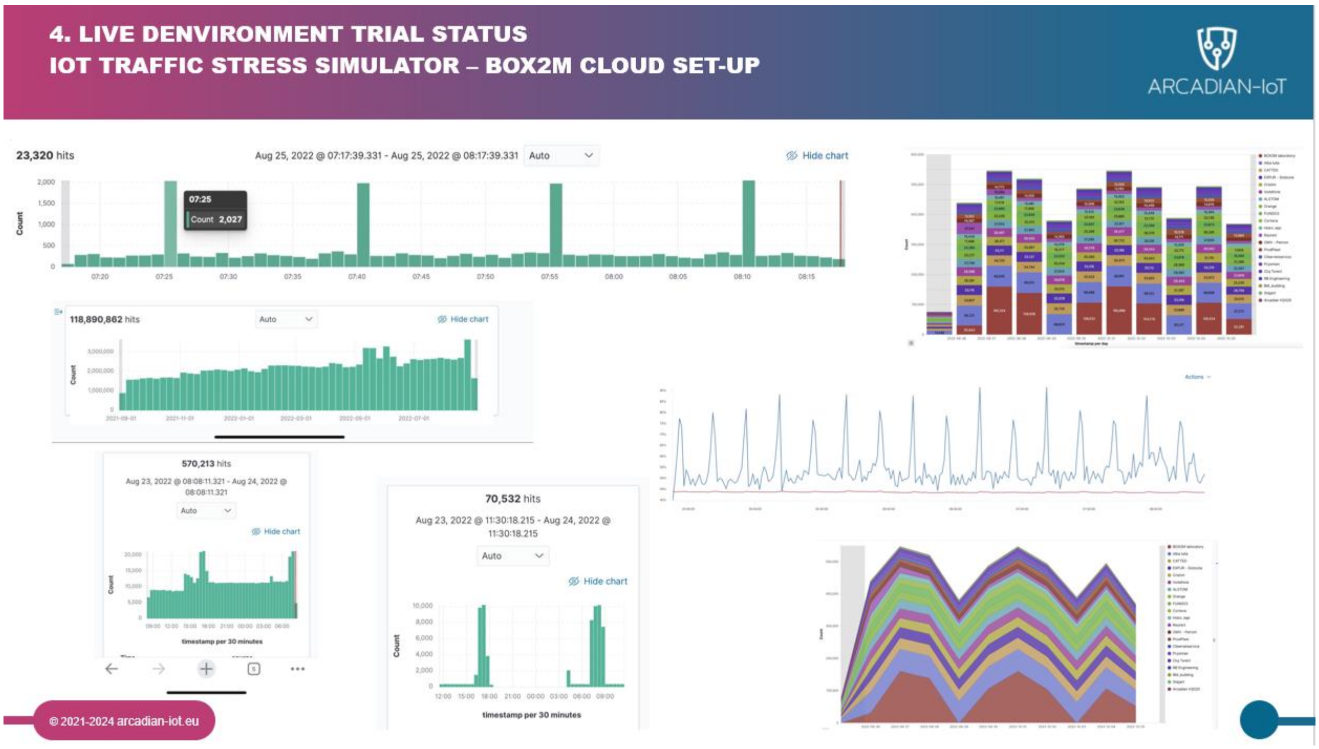

4. LIVE DENVIRONMENT TRIAL STATUS
IOT TRAFFIC STRESS SIMULATOR – BOX2M CLOUD SET-UP
ARCADIAN-IoT
23,320 hits
Aug 25, 2022 @ 07:17:39.331 - Aug 25, 2022 @ 08:17:39.331
Hide chart
07:25
Count 2,027
118,890,862 hits
570,213 hits
70,532 hits
timestamp per 30 minutes
© 2021-2024 arcadian-iot.eu

Figure 55 - IoT stress simulator

4.1.5 Future work

The HE component needs to be further developed in the following way, to satisfy the requirements of ARCADIAN-IoT platform and its use cases:

- The encryption library currently has a stable interface in Go and Python language, while it has also been tested with Java. The next step is to stabilize the interface in Java and extend the interface to C and JS due to the requirements of the use cases and other components.
- The encryption library needs to be extensively tested and benchmarked on various devices to prove the feasibility of the HE component and reach a KPI.
- The key management, that is currently providing basic functionalities needed for demonstrating the use of HE component, needs to be enhanced. In particular, the

integration with the blockchain component will allow to publish public cryptographic material in a decentralized way, while the integration with the authorization component will allow to delegate the private cryptographic keys to appropriate entities/devices. The component will also be used by the attestation component.

- The API of the key management authorities needs to be specified, while the functionality to obtain private and public cryptographic material needs to be implemented in the HE library
- The device middleware for the security applet needs to be adapted to personal devices. An Android smartphone with be target of research for this purpose.
- In WP5, solution providers will be involved in the reasoning for evaluating the performance of the HE component, particularly in what concerns the security applet. If relevant, research to try to optimize its performance will be done.
- For crypto chip system, it must be finished the firmware, in order to facilitate: the remote attestation system integration; the end-to-end encryption and decryption testing using 5G (by regular SIM and eSIM) - same as it was done for other boards; the encryption and decryption testing on OTA (over the air) scenarios, when condition monitoring grid scenarios will trigger sending of commands from IoT platform to device, through middleware, by relaying 2 types of encryptions (TLS and crypto chip key based); the self-recovery system integration; for crypto chip system, it must be finished the middleware, by finalizing integration with reputation system, remote attestation and self-aware data privacy system, In addition, it is foreseen the improvement of time KPIs described previously.

4.2 Permissioned Blockchain (ATOS)

4.2.1 Overview

4.2.1.1 Description

The Permissioned Blockchain is a common horizontal component in the ARCADIAN-IoT framework serving both Horizontal and Vertical Plane components such as Identity Management 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.

4.2.1.2 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 and persons.
- 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.

4.2.1.3 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 (M30)	***Current value (M20)***
3	0

KPI scope	
Deployment of permissioned blockchain in IoT environments	
Measurable Indicator	
Number of peer nodes deployed	
Target value (M30)	***Current value (M20)***
3	0

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 (M30)	***Current value (M20)***
20	0

4.2.2 Technology research

4.2.2.1 Background

ATOS does not have any background assets to build upon in providing the Permissioned Blockchain to meet the needs of ARCADIAN-IoT.

The State-Of-The-Art in Permissioned Blockchain technology is discussed in the following section.

4.2.2.2 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.

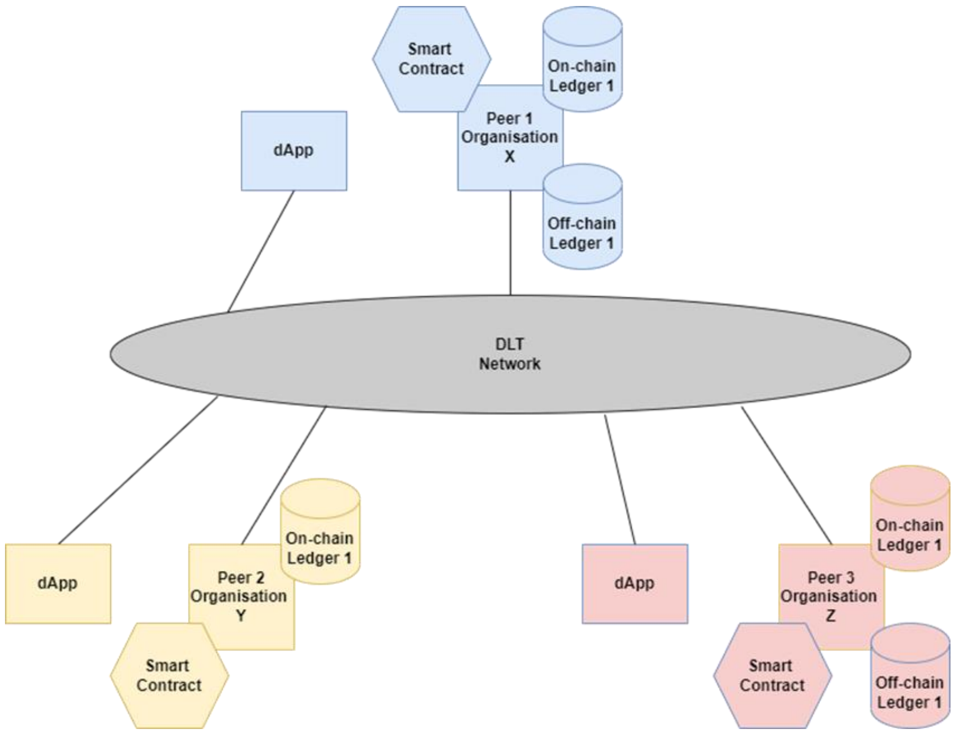


Figure 56 - Example of a blockchain network with three peer nodes.

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 blockchains will be deployed in three main components of the ARCADIAN-IoT framework: (i) Decentralized Identifiers, (ii) Reputation System, and (iii) Hardened Encryption.

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.

A high-level analysis of the state of the art of leading private Permissioned Blockchain technologies is carried out below.

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 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 [23] 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[27]

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[28]

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.

Quorum uses Solidity for smart contract developments; according to the DLT platforms comparison provided by Alastria,[29] 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[30]

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. Consensus is also handled in small groups of nodes (and then rolled up to the rest of the network) which means that transaction times per second are extremely low.

Smart Contracts are supported in the ABRA language (supported by extensions like TOQN) and run on QBriq. The implementation of tokenisation smart contract complexity is relatively limited when compared to Smart contracts languages (e.g., Solidity and DAML). The network is effectively centralized as a central coordinator is used to manage the network; however, IOTA 2.0 (Coordicide) with its fully decentralized approach, will substitute its previous version soon.

DLT Selection for ARCADIAN-IoT

[27] https://besu.hyperledger.org/
[28] https://github.com/ConsenSys/quorum
[29] https://alastria-es.medium.com/comparison-of-dlt-platforms-be84950d339d
[30] https://www.iota.org/

The evaluation approach in deciding which DLT to employ in the ARCADIAN-IoT framework 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, we have 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 on-chain; 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.

The methodology that can be used to evaluate Hyperledger Fabric smart contract transactions with benchmark figures is available on Hyperledger's GitHub.[31]

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.

4.2.2.3 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.

Decentralized Identifiers from Task 4.1 could likewise be published on the ARCADIAN-IoT Permissioned Blockchain, but the approach taken in Task 4.1 is to deploy them on a standard based Sidetree Node with the root anchor files hash stored on the blockchain, which is presently deployed on a Private Ethereum but could equally be deployed on a public blockchain.

4.2.3 Design specification

4.2.3.1 Sub-use cases

4.2.3.1.1 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.

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

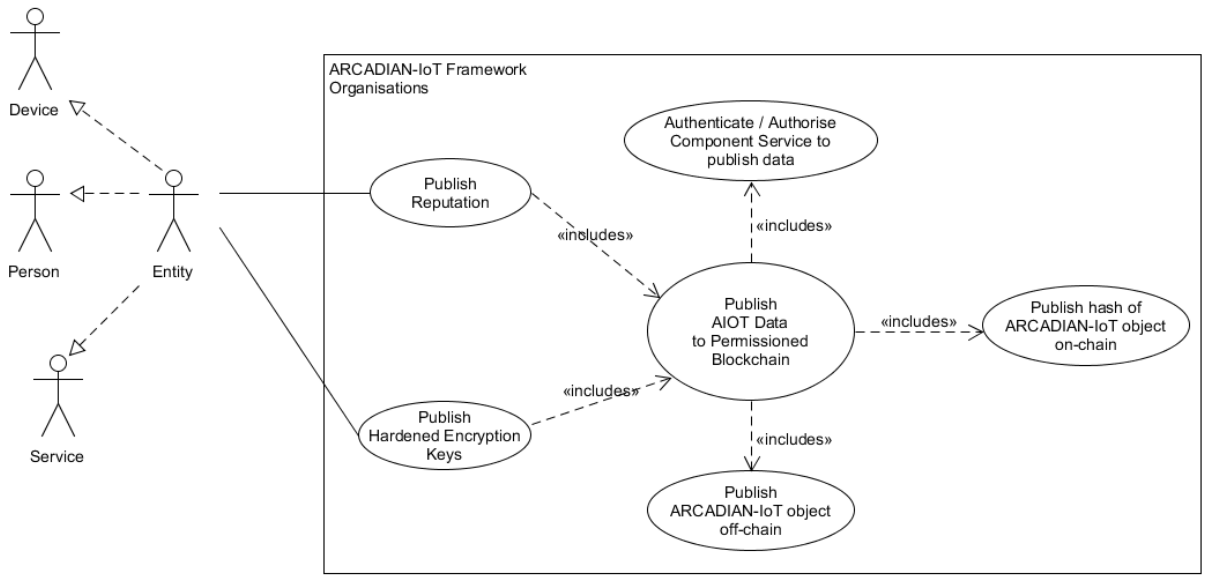

Device
Person
Entity
Service
ARCADIAN-IoT Framework Organisations
Authenticate / Authorise Component Service to publish data
Publish Reputation
«includes»
«includes»
Publish AIOT Data to Permissioned Blockchain
«includes»
Publish hash of ARCADIAN-IoT object on-chain
«includes»
Publish Hardened Encryption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

Figure 57 - 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.

4.2.3.1.2 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.

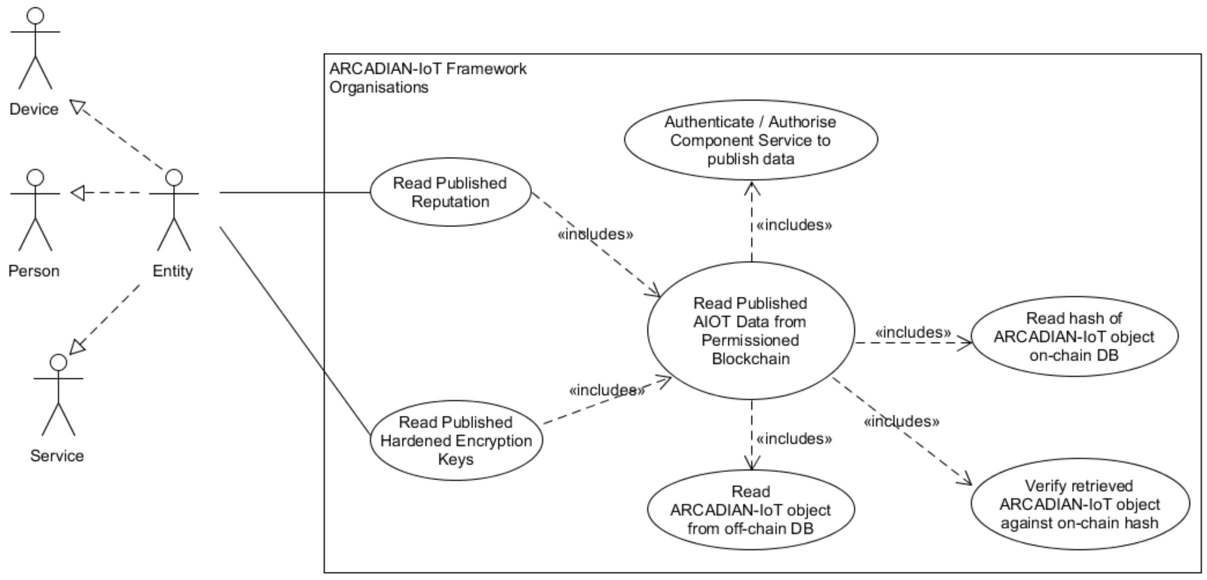

Device
Person
Entity
Service
ARCADIAN-IoT Framework Organisations
Authenticate / Authorise Component Service to publish data
Read Published Reputation
«includes»
«includes»
Read Published AIOT Data from Permissioned Blockchain
«includes»
Read hash of ARCADIAN-IoT object on-chain DB
«includes»
Read Published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

Figure 58 - 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.

4.2.3.1.3 Write and Fetch Sidetree blockchain transactions

To support public decentralized identifiers published on Sidetree as described in deliverable D4.2 [13] the hash of batched Sidetree DID operations are written to the blockchain so to provide a trust anchor for all DIDs published on a Sidetree Node. The primary blockchain transactions needed to support Sidetree are:

- Write a Sidetree transaction
- Fetch Sidetree transactions

This is depicted in the following use case diagram:

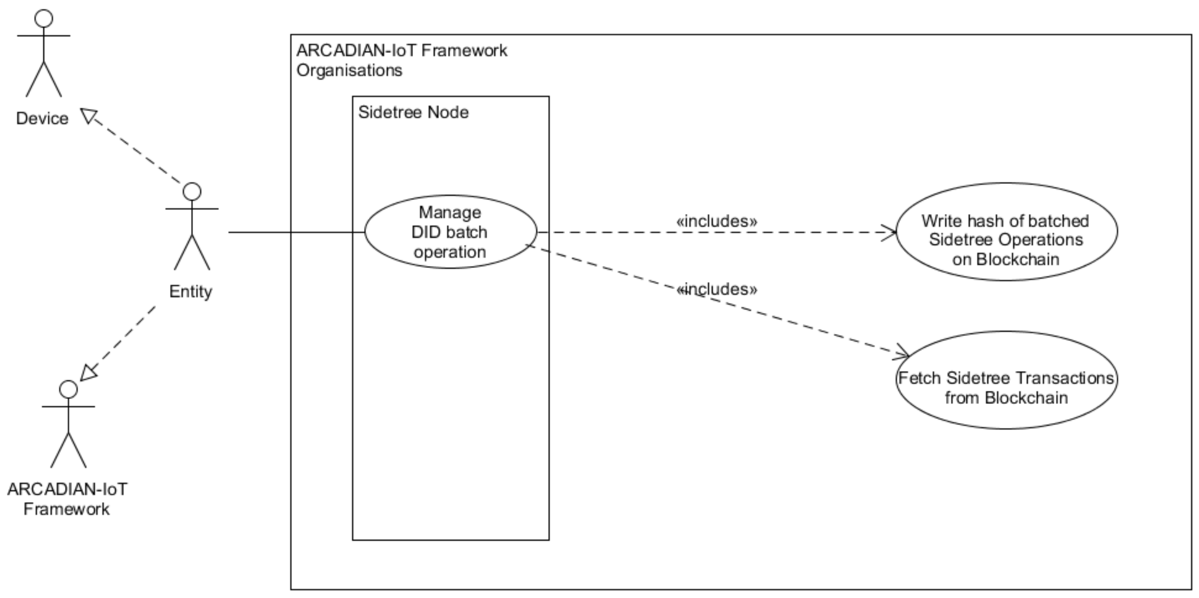

Device
Entity
ARCADIAN-IoT Framework
ARCADIAN-IoT Framework Organisations
Sidetree Node
Manage DID batch operation
«includes»
«includes»
Write hash of batched Sidetree Operations on Blockchain
Fetch Sidetree Transactions from Blockchain

Figure 59 - Write and Fetch Sidetree blockchain transactions

4.2.3.2 Logical architecture view

4.2.3.2.1 Publish ARCADIAN-IoT Data Objects

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

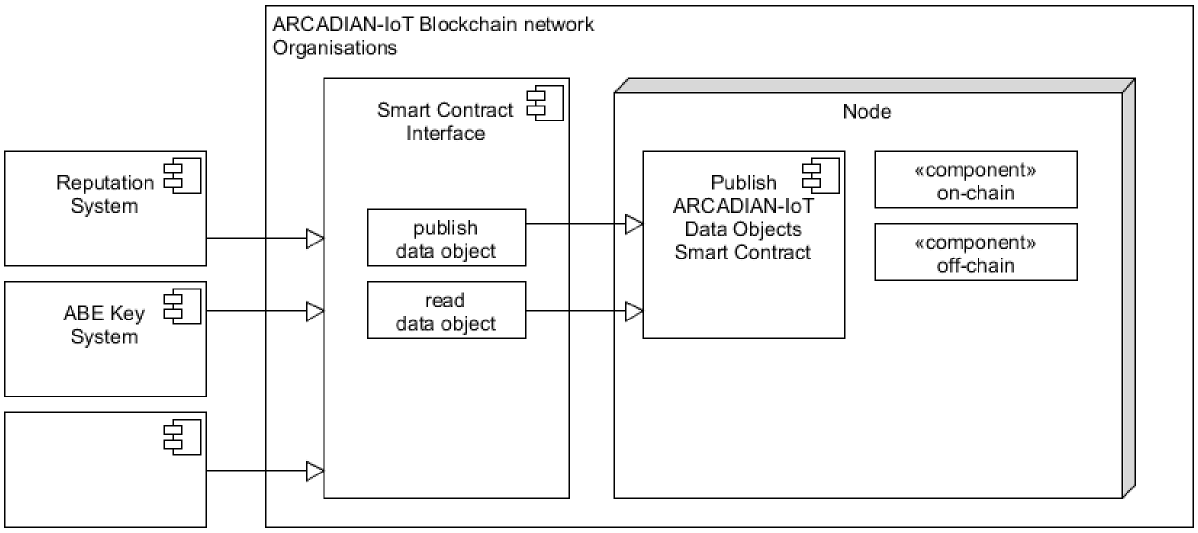


Figure 60 - Logical architecture for publishing ARCADIAN-IoT Objects to Permissioned Blockchain

Smart Contract Interface: This provides an interface for the ARCADIAN-IoT framework components to publish data to the permissioned blockchain. The Smart Contract interface supports HTTP Signature DID authentication [11] so that any ARCADIAN-IoT component can register their DID against the type of data object to be published so to allow it to be published.

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.

4.2.3.2.2 Publish Sidetree transactions

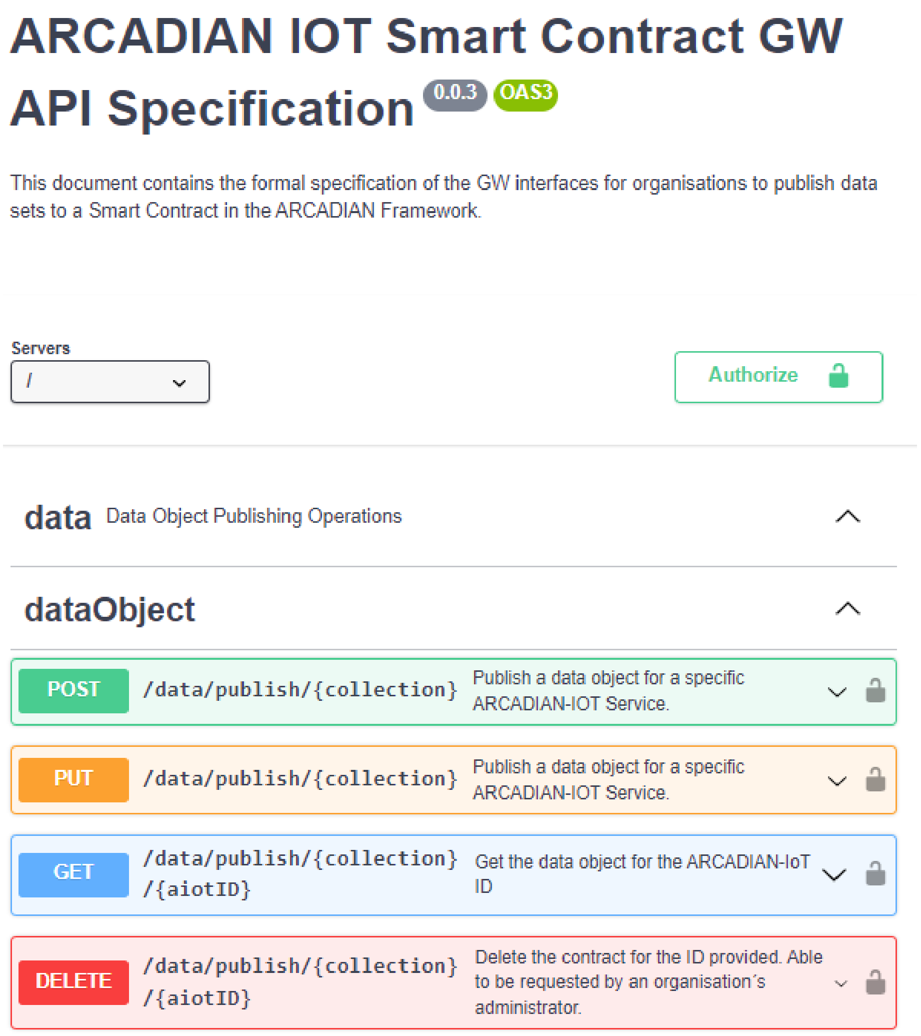
The Sidetree node, being implemented for the ARCADIAN-IoT framework to provide public DIDs, supports existing interfaces to Bitcoin and Ethereum public blockchains. The approach taken for the first prototype is to publish Sidetree transactions to Ethereum blockchain network deployed in a private network.

However, as the control of publishing of Sidetree transactions on the blockchain has no bearing on who is allowed to create Decentralized Identifiers in ARCADIAN-IoT there is little point in making this access permissioned. It is a matter of configuration for a Sidetree node to connect to the public Ethereum blockchain instead of the private Ethereum blockchain network.

The Sidetree protocol only publishes the hash of batched DID operations which can number tens of thousands of DID operations so this results in a very efficient use of the public blockchain, which is a main objective of Sidetree.

4.2.3.3 Interface description

4.2.3.3.1 Publish ARCADIAN-IoT Data Objects


ARCADIAN IOT Smart Contract GW API Specification 0.0.3 OAS3

This document contains the formal specification of the GW interfaces for organisations to publish data sets to a Smart Contract in the ARCADIAN Framework.

Servers: /

data Data Object Publishing Operations

dataObject

POST	/data/publish/{collection}	Publish a data object for a specific ARCADIAN-IOT Service.
PUT	/data/publish/{collection}	Publish a data object for a specific ARCADIAN-IOT Service.
GET	/data/publish/{collection}/{aiotID}	Get the data object for the ARCADIAN-IoT ID
DELETE	/data/publish/{collection}/{aiotID}	Delete the contract for the ID provided. Able to be requested by an organisation´s administrator.

Figure 61 - Publish ARCADIAN-IoT Data Objects Interface Description

4.2.3.3.2 Publish Sidetree transactions

The Sidetree transaction interface with the blockchain is integrated with the Sidetree node implementation as described under Decentralized Identifier component in deliverable D4.2 [13].

4.2.3.4 Technical solution

4.2.3.4.1 Deployment architecture view

Publishing ARCADIAN-IoT Data deployment view

The Permissioned Blockchain deployment architecture is based on Hyperledger Fabric 4.2. 3 Peer Nodes will be deployed for supporting the publishing of ARCADIAN-IoT Data Objects as shown in the figure below.

There is one Smart Contract Interface per ARCADIAN-IoT framework deployment and external organisations can also be registered onto the blockchain network and configured with read / write permissions to the blockchain, as per the Membership Service Provider provided by Hyperledger Fabric based on X.509 certificates.

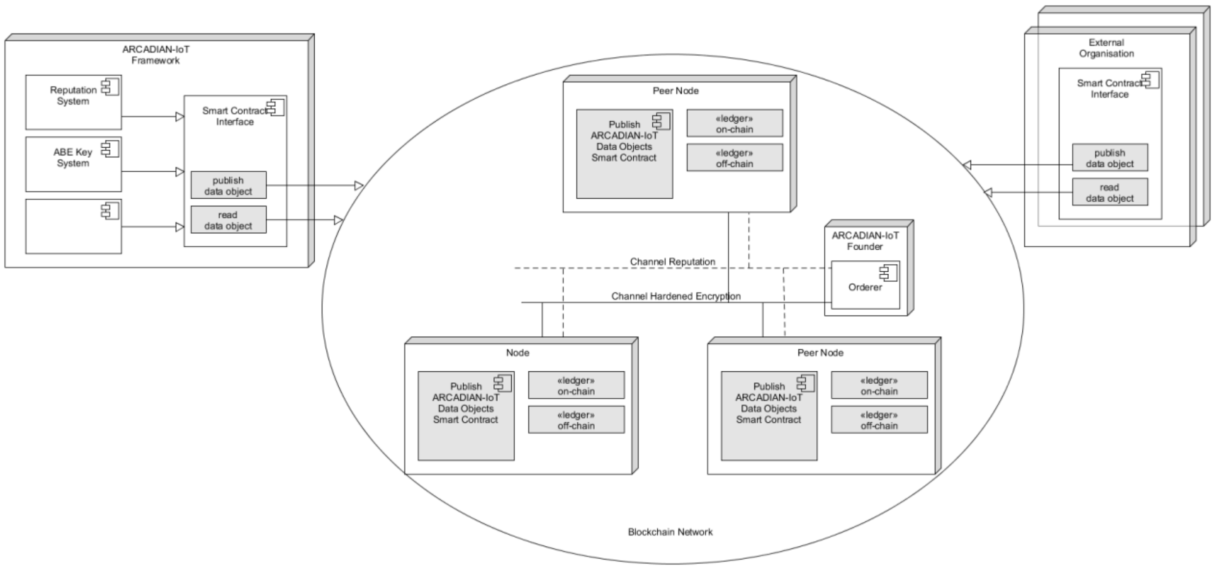


Figure 62 - Permissioned Blockchain for Publishing ARCADIAN-IoT Data deployment view

Publishing Sidetree transactions deployment view

Please refer to deliverable D4.2 - Vertical Planes [13] for the overall deployment view of the blockchain network supporting the Sidetree node.

4.2.3.4.2 API specification

Publishing ARCADIAN-IoT Data deployment view

The OpenAPI YAML specification is published on the ARCADIAN-IoT GitLab repository here: https://gitlab.com/arcadian_iot/blockchain/-/blob/main/interface/openapiPB.yml

Publishing Sidetree transactions deployment view

Please refer to deliverable D4.2 - Vertical Planes [13].

4.2.3.4.3 Smart Contract GW Interface design

The Smart Contract GW Interface implements the Open API interface to provide API requests to publish ARCADIAN-IoT data objects to the ARCADIAN-IoT Permissioned Blockchain without the publisher needing to implement any specific Hyperledger Fabric technology to carry out the transactions. See previous section for the full API specification for publishing data objects to the blockchain.

ARCADIAN-IoT APIs are protected by HTTP Signature performing authentication of the source of the request that it is authorised to access the GW based on their Decentralized Identifier. The source may be the ARCADIAN-IoT Framework components or external 3rd parties.

The table below demonstrates an example for the DID Publishing authorisation permissions that are configured in a JSON file.

DID	Business Object	Write	Read
`did:web:example.com:comp:reputation`	Reputation	YES	YES

`did:web:example.com:service:auditor`	Reputation	NO	YES
`did:web:example.com:service:auditor`	abeKey	NO	YES

Table 16 - ARCADIAN-IoT Smart Contract GW Interface Authorisation Permissions

4.2.3.4.4 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 or 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.

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

The access to Private Data Collections are controlled by:

- The member organisations given permission to take part in that blockchain channel
- A collection definition 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.

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[32].

Data Model

For publishing Reputation data an off-chain Private Data Collection specific to the type of

[32] https://hyperledger-fabric.readthedocs.io/en/latest/couchdb_tutorial.html

Reputation data object to be published is created, as follows:

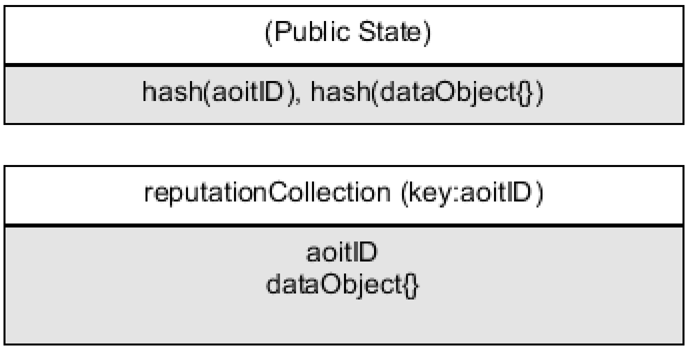

(Public State)
hash(aoitID), hash(dataObject{})
reputationCollection (key:aoitID)
aoitID
dataObject{}

Figure 63 - 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:

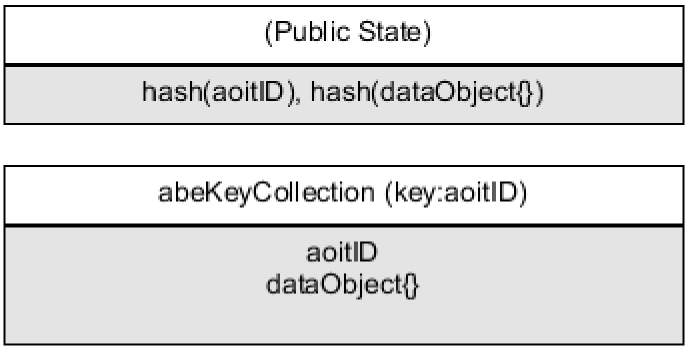

(Public State)
hash(aoitID), hash(dataObject{})
abeKeyCollection (key:aoitID)
aoitID
dataObject{}

Figure 64 - ABE Key Private Data Collection

A channel´s Private Data Collections is able to be validated against the hash stored on the channels on-chain world state as well as to be completely purged [17].

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 commited 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 uses 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].

4.2.3.4.5 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.

4.2.4 Evaluation and results

The smart contract implementation is currently nearing completion of development and test in a local environment.

4.2.5 Future work

The next step is to integrate and test the publishing of Reputation and Hardened Encryption data first against a local deployment of the ARCADIAN-IoT Data Smart Contract deployed on Hyperledger Fabric and then to deploy on piloting partners´ infrastructure in collaboration with WP5.

As regards, support of the Ethereum deployment to support DIDs published over a sidetree node this is deployed firstly in a local deployment for testing DID creation for an SSI Agent as part of Decentralized identifier component integration. Once integration is complete, it can be deployed as a Private Ethereum on piloting partners infrastructure or use the public Ethereum blockchain.

For the final prototype, it is also planned to also explore replacing the private Ethereum with the Hyperledger Fabric as the blockchain to store the anchor file hashes and make use of ARCADIAN-IoT´s deployed Permissioned Blockchain.

5 CONCLUSIONS

In the present intermediate deliverable, the partners involved in WP3 (IPN, ATOS, MAR, RISE, BOX2M, UWS, XLAB, and TRU) have reported on the current status of the components in the Horizontal Planes of the ARCADIAN-IoT framework: 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, and Permissioned Blockchain.

In the reporting period, the partners involved continued the technological research of each component and provided their descriptions. The initial design provided in the deliverable D3.1 has been extended, and its implementation is in process. The description of the current resources, prototypes and results given in this deliverable will be crucial for the use cases in WP5 to start their integration. Furthermore, detailed design specifications for each component have been provided, explaining the internal workflows and interfaces to the other ARCADIAN-IoT components. This presents a major step in developing the platform as an interconnected unity of many subcomponents.

The following steps in the development of the ARCADIAN-IoT framework have been described and the results of it will be reported in the final WP3 deliverable D3.3.

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] 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 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.
18 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.
19 Perazzo, Pericle, et al. "Performance evaluation of attribute-based encryption on constrained iot devices." Computer Communications 170 (2021): 151-163.

20 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.

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] 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.

[23] 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/

[13] ARCADIAN-IoT, “D4.2 - Vertical Planes”, 2022

[14] Private Data Collections tutorial. https://hyperledger-fabric.readthedocs.io/en/latest/private_data_tutorial.html , 2022

[15] Peer Lifecycle Chaincode, https://hyperledger-fabric.readthedocs.io/en/latest/commands/peerlifecycle.html#peer-lifecycle-chaincode-commit, 2022

[16] Hyperledger Fabric Chaincode APIs, https://hyperledger.github.io/fabric-chaincode-node/main/api/fabric-shim.ChaincodeStub.html#getPrivateData__anchor, 2022

[17] https://hyperledger-fabric.readthedocs.io/en/release-1.3/private-data/private-data.html, 2022

[18] https://hyperledger-fabric.readthedocs.io/en/release-2.4/chaincode4ade.html, 2022

[19] ARCADIAN-IoT, “D2.4 ARCADIAN-IoT framework requirements”, 2021

[20] https://hyperledger-fabric.readthedocs.io/en/release-2.4/whatis.html, 2022