



PRIVATEER

Privacy-first Security Enablers
for 6G Networks

Deliverable 2.3

PRIVATEER Framework Demonstrator – Rel. A

DRAFT – Pending approval by the Smart Networks and Services Joint Undertaking (SNS JU)



PRIVATEER has received funding from the Smart Networks and Services Joint Undertaking (SNS JU) under the European Union's Horizon Europe research and innovation programme under Grant Agreement No. 101096110

Space Hellas SA, NCSR "Demokritos, Telefónica I&D, RHEA System SA, INESC TEC, Infil Technologies PC, Ubitech Ltd, Universidad Complutense de Madrid, Institute of Communication and Computer Systems, Forsvarets Forskningsinstitut, Iquadrat Informatica SL, Instituto Politecnico do Porto, ERTICO ITS Europe



PRIVATEER

Deliverable 2.3

PRIVATEER framework demonstrator – Rel. A

Deliverable Type
Report/Demonstrator

Month and Date of Delivery
July 19th 2024

Work Package
2

Leader
NCSR

Dissemination Level
Public

Authors
Georgios Xylouris (NCSR), Maria
Christopoulou (NCSR) and Dimitris
Santorinaios (NCSR)

Programme
Horizon Europe

Contract Number
101096110

Duration
36 months

Starting Date
January 2023

Contact Us

privateer-contact@spacemaillist.eu



PRIVATEER has received funding from the Smart Networks and Services Joint Undertaking (SNS JU) under the European Union's Horizon Europe research and innovation programme under Grant Agreement No. 101096110



Contributors

Name	Organization
Anna Angelogianni, Nikos Fotos, Thanassis Giannetsos, Manos Kalotychos	UBITECH
Mattin Elorza, Antonio Pastor	TID
Pedro Sousa, António Pinto, João Vilela	INESC TEC
Ilias Papalabrou, Dimosthenis Masouros, Aimilios Lefteriotis	ICCS
Elmira Saeedi, Jesús A. Alonso-López	UCM
Dimitris Santorinaios, Maria Christopoulou	NCSR
Apostolos Garos, Georgios Gardikis	SPH
Anastasios Bikos	IQUADRAT
Fábio Silva, Ricardo Santos	IPP
Lampros Argyriou, Antonia Karamatskou	INFILI

Reviewers

Name	Organization
Georgios Gardikis	SPH
Cristian Petrollini, César Peñacoba	RHEA/STARION



Copyright and Disclaimer

This document may not be copied, reproduced or modified in whole or in part for any purpose without written permission from the Editor and all Contributors. In addition to such written permission to copy, reproduce or modify this document in whole or part, an acknowledgement of the authors of the document and all applicable portions of the copyright notice must be clearly referenced.

The information in this document is provided “as is”, and no guarantee or warranty is given that the information is fit for any particular purpose. The reader uses the information at his/her sole risk and liability. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or SNS JU. Neither the European Union nor the granting authority can be held responsible for them



Version History

Version	Date	Modifications
1.0	19/7/2024	First Version



List of Acronyms

Acronym	Description
ABAC	Attribute-Based Access Control
ABE	Attribute-Based Encryption
ADMM	Alternating Direction Method of Multipliers
API	Application Programming Interface
ATL	Actual Trust Level
B5G	Beyond 5G
BIOS	Basic Input/Output System
CI/CD	Continuous Integration (CI) Continuous Delivery (CD)
CIV	Configuration Integrity Verification
CPU	Central Processing Unit
CTI	Cyber Threat Intelligence
DB	DataBase
DDoS	Distributed Denial of Service
DDPG	Deep Deterministic Policy Gradient
DDR	Double Data Rate
DID	Decentralized Identifier
DNS	Domain Name System
DLT	Distributed Ledger Technology
DP	Differential Privacy
DQN	Deep Q-Networks
DRAM	Dynamic Random-Access Memory
E2E	End-to-end
eBPF	extended Berkeley Packet Filter
FL	Federated Learning
FPGA	Field-Programmable Gate Array
gNB	Next Generation Node B
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
HPA	Horizontal Pod Autoscaler
IBNSC	Intent-Based Networking Services Catalogue
ICMP	Internet Control Message Protocol
IoC	Indicator of Compromise
IMEISV	International Mobile station Equipment Identity Software Version
IP	Internet Protocol
LoT	Level of Trust
LSTM	Long Short-Term Memory
MISP	Malware Information Sharing Platform
MiTM	Man in The Middle
MNO	Mobile Network Operator



MPC	Multi-Party Computation
NWDAF	Network Data Analytics Function
NAS	Network Attached Storage
NFV	Network-Function Virtualization
OCI	Open Container Initiative
OPoT	Ordered Proof of Transit
PCIe	Peripheral Component Interconnect Express
PI	Privacy Index
PoC	Proof of Concept
PoT	Proof of Transit
RAN	Radio Access Network
RoT	Root of Trust
SC	Smart Contract
SCB	Security Context Broker
SDN	Software Defined Network
SD-WAN	Software-Defined Wide-Area Network
SGX	Software Guard Extensions
SLA	Service Level Agreement
SGC	Service Graph Chain
SP	Service Provider
SOAR	Security Orchestration Automation & Response
SSI	Self-Sovereign Identities
TEE	Trusted Execution Environment
TLS	Transport Layer Security
UE	User Equipment
UUID	Universally Unique IDentifier
VNF	Virtual Network Function
vRAN	Virtualized Radio-Access Networks (vRAN)
VP	Verifiable Presentation
XAI	Explainable Artificial Intelligence



Executive Summary

The PRIVATEER project aims to cover the challenges of securing B5G networks through a framework that integrates advanced security mechanisms and privacy enablers. The project's mission is to develop a comprehensive framework that ensures trust, security, and privacy across the infrastructure. This document describes the first integrated release of the PRIVATEER framework. The release is based on the work carried out by the consortium to implement the architecture for the PRIVATEER as defined D2.2 [1]. Furthermore, it outlines the functionalities already implemented to achieve the initial objectives of the project and the specific use case requirements. To that end, the project has designed three workflows that verify the integration between various components, ensuring that they interact efficiently and effectively to form a cohesive system. Each workflow highlights specific functionalities of the components deployed as part of this release and demonstrates the integration between these components. These workflows, which directly correspond to -and are part of- the use cases/storylines described in Deliverable D2.2, are as follows:

Workflow 1: Enhanced Security for Virtualized Infrastructures

This workflow focuses on protecting data in use through Trusted Execution Environments (TEEs). It employs Security Probes and μ Probes to collect and verify attestation evidence, ensuring the integrity of containerized applications and the underlying infrastructure. The process involves secure deployment, continuous monitoring, and real-time trust assessments, facilitated by the SCHEMA Privacy-aware Orchestrator and PRIVATEER Blockchain.

Workflow 2: DDoS Detection and CTI Sharing

This workflow addresses DDoS attacks using a federated learning (FL) model for anomaly detection. The security analytics module installed on edge nodes utilizes the NWDAF to monitor network traffic and detect attacks. Detected anomalies trigger alerts and engage the CTI-sharing proxy API for threat intelligence sharing. This approach ensures coordinated responses to DDoS attacks and enhances overall network security.

Workflow 3: Real-Time Mitigation of Man-in-the-Middle Attacks

In this workflow, for real-time mitigation of Man-in-the-Middle (MiTM) attacks, the Proof of Transit (PoT) method is employed. This involves deploying a Software Defined Network (SDN) and configuring nodes to verify packet paths. The workflow ensures that any packet not following the trusted nodes is discarded, thereby preventing unauthorized data interception and modification.



Table of Contents

1	Introduction	11
1.1	Purpose of the document	11
1.2	Relation to other project work	11
1.3	Structure of the document	11
2	PRIVATEER Architecture	13
3	Integration Methodology	16
3.1	GitHub	16
3.2	Harbor	17
3.3	Slack	17
4	Integration Environment	18
5	Workflow 1: Enhanced Security for Virtualized Infrastructures & Level of Trust Assessment	20
5.1	Attack models	20
5.2	Workflow 1 implementation.....	20
5.2.1	Short Description	23
5.2.2	Detailed Description	24
5.2.3	APIs/interfaces	29
5.2.4	Preliminary Results	44
5.3	FPGA attestation	47
5.3.1	APIs – Interfaces.....	50
5.3.2	Preliminary Results	51
6	Workflow 2: DDoS detection by Federated NWDAF & CTI Sharing and SLA Management.....	57
6.1	Attack models	57
6.2	Workflow 2 implementation.....	60
6.2.1	Short Description	60
6.2.2	Detailed Description	61
6.2.3	APIs/Interfaces	63
6.2.4	Preliminary Results	67
6.2.5	Workflow 2 Demonstrators	67
7	Workflow 3: Real-Time Mitigation of Man in the Middle Attacks	76



7.1	Attack models	76
7.2	Workflow 3 implementation.....	76
7.2.1	Short Description	76
7.2.2	Workflow.....	76
7.2.3	APIs/Interfaces.....	79
7.2.4	Preliminary Results	79
8	Conclusion & next steps	83
	References	84
Annex A:	Mapping of Workflows to PRIVATEER Use Cases	85



1 Introduction

1.1 Purpose of the document

This document outlines the details of the PRIVATEER framework alpha release, a collaborative effort under Work Package 2, integrating the components developed under technical work packages 3 to 5. The primary aim is to explain the solutions included in this initial release, the interaction between components, the functionalities provided, and the future plans for the project. This gives the reader a comprehensive understanding of the alpha release and its capabilities. The document details the architectural changes and current functionalities, illustrated through three workflows. Additionally, it describes the development process and the project's approach to continuous integration and delivery.

1.2 Relation to other project work

This deliverable outlines the results of the architecture definition and updates to the design previously presented in "D2.2 Use Cases, Requirements, and Design Report" [1]. It represents the first of two releases planned for the project, preceding the second release (D2.4) scheduled for month 36 of the project.

As the PRIVATEER first release incorporates results from all technical work packages, this document is closely connected to deliverables D3.1 [2], D4.1 [3], and D5.1 [4], which provide detailed descriptions of the PRIVATEER components. To avoid redundancy, this document offers a high-level overview, and readers are encouraged to refer to the mentioned documents for detailed information.

1.3 Structure of the document

The document is structured in 8 chapters:

Chapter 1. *Introduction*: (this section) presents the objectives of the document and introduces its content, structure, and relationships with other project's documents.

Chapter 2. *PRIVATEER Architecture*: describes the updated architecture, including key components like the Blockchain Layer, the new Intermediary Layer, and the Privacy Aware Orchestration layer.

Chapter 3. *Integration Methodology*: explains the tools and resources used to support development and integration.

Chapter 4. *Integration Environment*: provides a description of the integration environment.



Chapters 5-7 (*Workflow 1-3*): describe the attack models mitigated by each workflow, the workflow implementation, the components and interfaces involved and preliminary evaluation results.

Chapter 8. *Conclusion & Next Steps* provides a summary of the current release's achievements and an outline of the project's future plans and next steps.

2 PRIVATEER Architecture

The PRIVATEER framework high-level architecture has been updated, as depicted in Figure 1, based on the design decisions and findings stemming from the progressing work packages (WP3/4/5). The key updates and the reasons behind them are as follows:

Blockchain Layer:

The blockchain layer introduced has been divided into two components:

Distributed Ledger Technology (DLT) Off-Chain: This component stores trustworthiness evidence, allowing for efficient handling and verification of trust-related data without overloading the main blockchain. The decision to store information off-chain is influenced by factors such as the payload size. For instance, attestation evidence might be optimally stored off-chain, while maintaining integrity and authenticity checks on-chain to prevent tampering with the off-chain data.

DLT On-Chain: This component manages identity-related information and the quantification of the Level of Trust (LoT). By separating identity management and trust quantification, we ensure better organization and security of sensitive information.

A new intermediary layer has been introduced between the DLT components and the other parts of the framework. This layer is responsible for forwarding and translating information between the DLT and the rest of the system. It consists of two key components, i) the **Security Oracle**, and ii) the **Security Context Broker**.

The i) **Security Context Broker (SCB)** facilitates the creation of smart contracts for the Secure Oracle. It provides the list of attributes, Privacy Service Level Agreement (SLA), service graph chain topology, and Trust Policies received from the Orchestrator and LoT Assessment component. It additionally facilitates the task of retrieving information from the ledger for the LoT Assessment component and the Orchestrator, meaning that the LoT and the Orchestrator may access information stored on the DLT through the Security Context Broker.

The ii) **Secure Oracle** operates with the support of a Trusted Execution Environment (TEE); thus, acts as an integral part of the PRIVATEER Trusted Computing Base (TCB). As such, it is inherently considered a trusted component within the system. It performs tasks such as receiving and validating attestation reports from Security Probes and generating smart contracts with comprehensive information for trust assessment processes. These contracts include Privacy Service Level Agreements (SLAs), information regarding the service graph, trust policies, trust parameters, access control attributes, and pointers to off-chain storage for (failed) attestation evidence. Smart contracts are signed with the Secure Oracle's key, allowing any entity accessing the Ledger to verify their authenticity. Whenever a new trust evaluation

occurs, the outcome of this assessment can be incorporated into the smart contract. This integration occurs through communication between the LoT Assessment component and the Secure Oracle, allowing for the seamless inclusion of the evaluation result within the smart contract.

Trust Exposure Layer:

The Trust Exposure Layer is a mechanism that protects privacy by limiting the information provided to external entities like Mobile Network Operators (MNOs), users, or Service Providers. This layer ensures that the information provided is limited to the trust state of a service, including trust property and level of trust. It removes details (i.e., evidence) that may leak information about the underlying infrastructure or service graph, protecting privacy without compromising external entities' ability to assess the trustworthiness of microservices. This harmonisation mechanism ensures that only relevant and necessary information is shared, effectively mitigating risks.

Privacy Aware Orchestration:

The Privacy Aware Orchestration layer has been extended to include the LoT Controller, which collects information about the end-to-end (E2E) service and estimates its trustworthiness. The Security Orchestration, Automation & Response (SOAR) component is added, which, combined with the Decision Engine and utilizing metrics from the LoT and the Privacy Index (PI), makes informed decisions about the service lifecycle. Another highlighted component is the intent-based networking manager, which is composed of the Intent-Based Networking Services Catalogue (IBNSC), the SLA Manager, and the Smart Contracts templates. The IBNSC will be populated with a set of services that utilize available network resources efficiently. The SLA Manager tracks each established service and its specifications, ensuring that service level agreements are maintained. The smart contracts streamline deployment by providing templates for Intent-Based Services, ensuring consistency and efficiency in contract creation and execution.

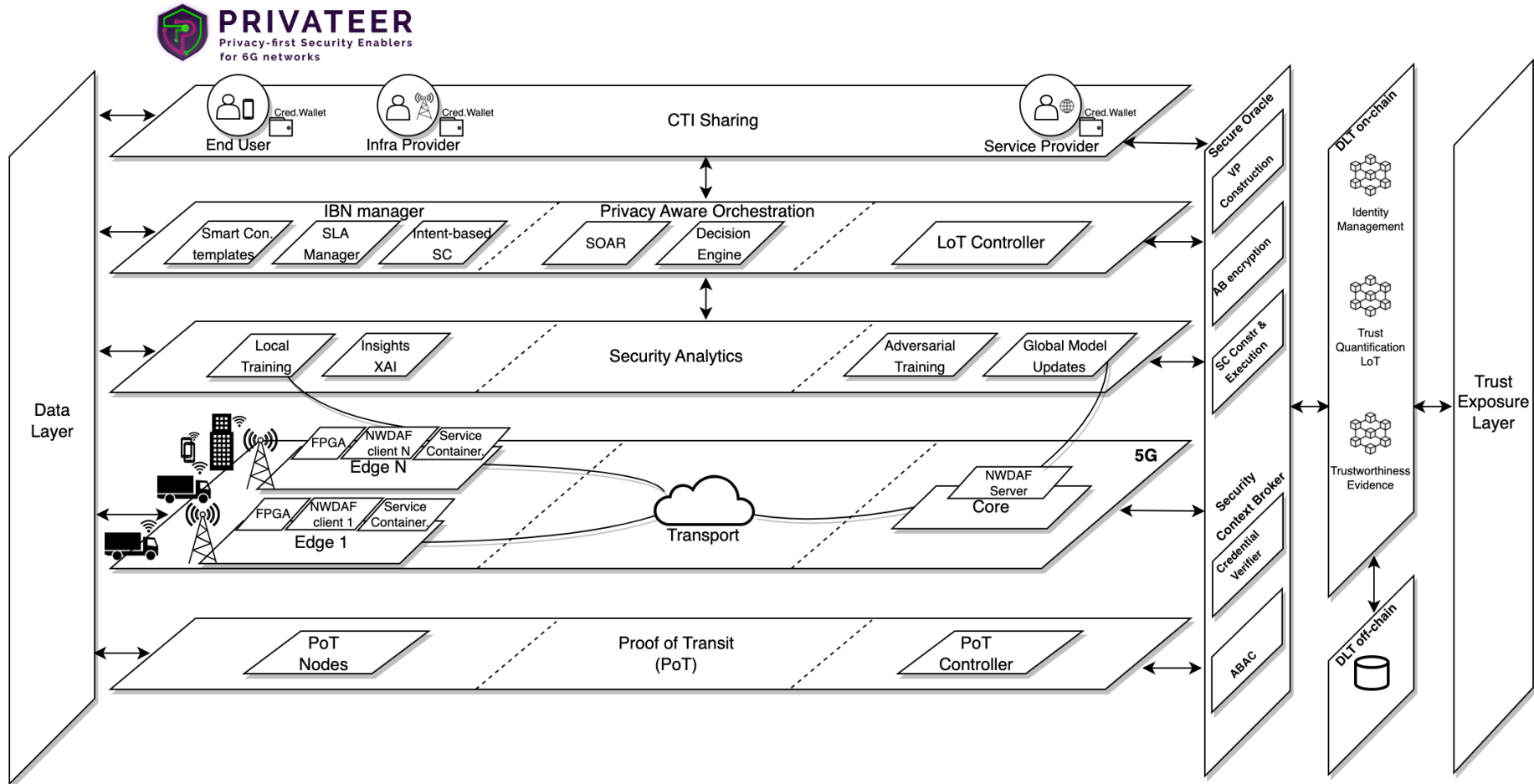


Figure 1 - Updated PRIVATEER High-level architecture

3 Integration Methodology

To support the development and integration processes, PRIVATEER has established an infrastructure composed of various resources and tools, which enables the collaborative effort of the consortium partners. Figure 2 below illustrates the components of this infrastructure as of June 2024. The plan is to continually assess and incorporate additional tools and integrations to meet the evolving requirements for integrating and testing future releases.

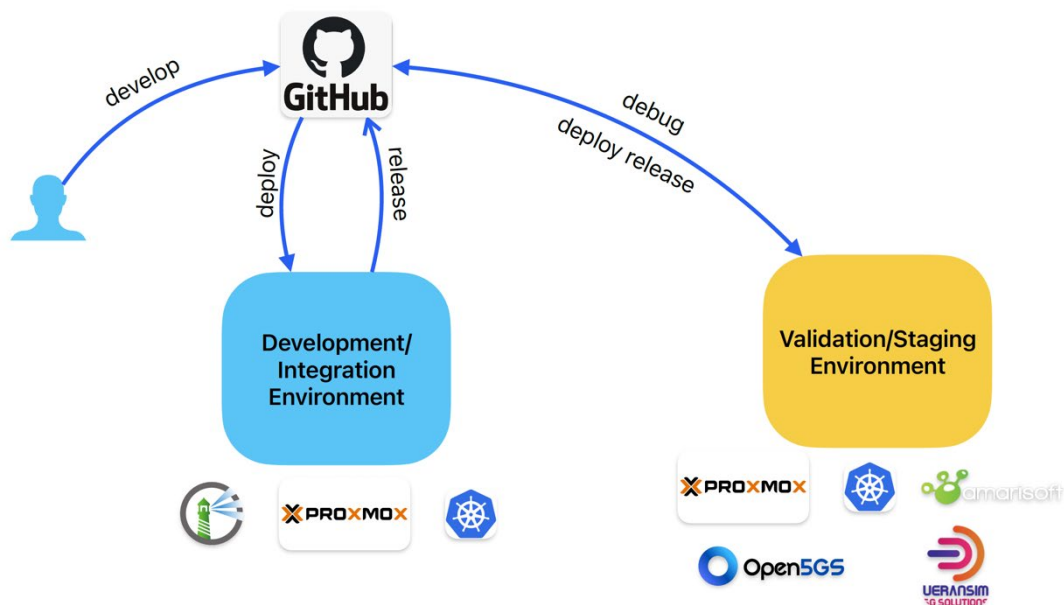


Figure 2 - Integration Environment

Once requested, users needing to access the infrastructure receive a username and password that can be used to access the infrastructure. OpenVPN has been used to establish secure and encrypted connections for remote access. User accounts have been configured to ensure secure remote administration, monitoring, and access to the environment. The following subsection briefly presents the tools, explaining why and how they are used in the project.

3.1 GitHub

GitHub is the main code repository platform for the project, accessible at <https://github.com/privateer-project>. It provides several tools used in the project:

Source code repositories

GitHub stores the source code and technical documentation for the software components developed in the PRIVATEER project.



Collaborative development

GitHub includes several features to help developers work together:

- **WEB IDE:** Allows file changes directly from the GitHub interface.
- **Branches and Merge Requests:** Lets collaborators suggest changes to the code in a controlled manner.
- **Roles:** Assigns different permissions to different people.
- **Issue Trackers and To-Do Lists:** Helps manage, assign, and discuss technical tasks.
- **Tags and Releases:** Creates references to specific versions of the code.

Technical Documentation

GitHub is used to store the API definitions (usually in OpenAPI format) and technical documentation for each component. This helps other project developers who need to use or integrate with the component. Keeping all this information in one place and in a standard format makes it easier to find and reduces communication time. Currently, there are no strict requirements for the format and type of documentation needed for each component, but clearer rules will be established for future PRIVATEER releases.

3.2 Harbor

Harbor registry is planned to be used to manage Docker images. Harbor is an open-source Open Container Initiative (OCI) registry which can be used privately, ensuring secure storage and access to our container images. It includes image signing and vulnerability scanning, enhancing our security posture. Additionally, Harbor can be integrated with GitHub CI/CD pipelines, simplifying the development and deployment processes, and ensuring efficient version control and reliable distribution of containerized applications across the project.

3.3 Slack

Slack is a collaboration tool that helps the communication between development teams. It makes discussions quicker and accessible to all developers. It has been chosen for its ease of use and integration capabilities. Slack supports one-to-one conversations, group chats, and channels for unlimited members. Channels can be created per topic to keep related discussions organized. Predefined channels have been created for each work package to facilitate work package-wide conversations. Additional channels can be created by any member to discuss specific topics. Moreover, plugins can be added to integrate Slack with other automation tools, such as notifications for GitHub events, deployment events, email notifications and so forth.

4 Integration Environment

The infrastructure provided by NCSR D, shown in Figure 3, is designed to support diverse computing and networking requirements, such as virtualization, container orchestration, and secure remote access. It features a Proxmox¹ cluster of five dedicated physical servers, utilizing the Proxmox Virtual Environment (Proxmox VE) to manage virtual machines and containers. This cluster supports the PRIVATEER components that are combined (see Figure 4) to provide the first release. These servers are connected via a Gigabit Ethernet network switch. Additionally, a centralized 20TB network-attached storage (NAS) solution is integrated into the cluster. This storage is crucial for data sharing and redundancy, ensuring that virtual machines and containers can access and store data across the network. Furthermore, the infrastructure includes a 5G system with two Next Generation Node B (gNBs) positioned at the edge. Both gNBs are connected to the core network and support seamless handover for User Equipment (UE) devices.

¹ <https://www.proxmox.com/en/>

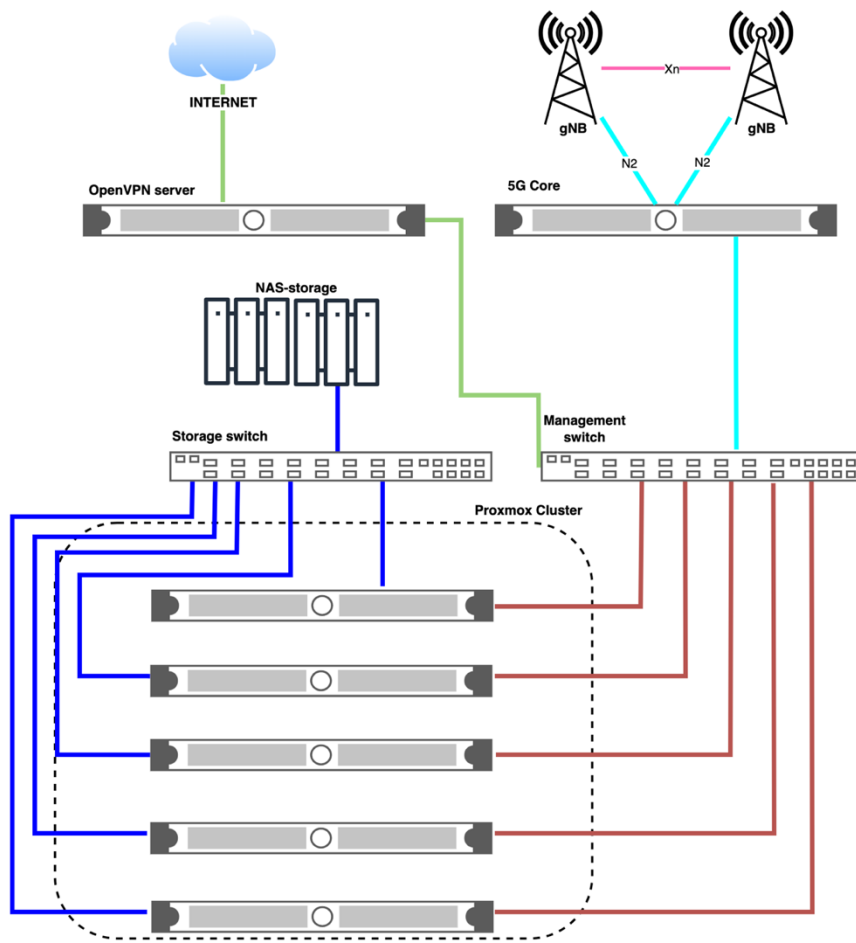
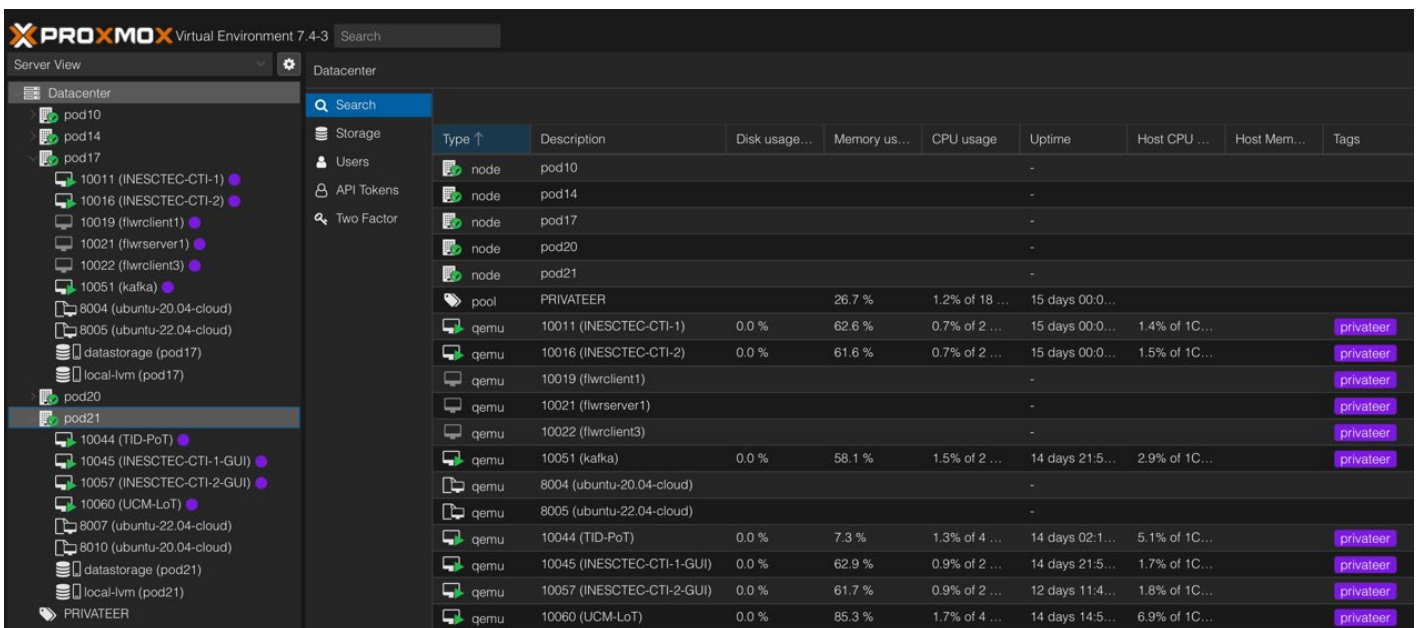


Figure 3 - Testbed diagram



Type	Description	Disk usage...	Memory us...	CPU usage	Uptime	Host CPU ...	Host Mem...	Tags
node	pod10							
node	pod14							
node	pod17							
node	pod20							
node	pod21							
pool	PRIVATEER		26.7 %	1.2% of 18 ...	15 days 00:0...			
qemu	10011 (INESCTEC-CTI-1)	0.0 %	62.6 %	0.7% of 2 ...	15 days 00:0...	1.4% of 1C...		privateer
qemu	10016 (INESCTEC-CTI-2)	0.0 %	61.6 %	0.7% of 2 ...	15 days 00:0...	1.5% of 1C...		privateer
qemu	10019 (flwrclient1)							privateer
qemu	10021 (flwrserver1)							privateer
qemu	10022 (flwrclient3)							privateer
qemu	10051 (kafka)	0.0 %	58.1 %	1.5% of 2 ...	14 days 21:5...	2.9% of 1C...		privateer
qemu	8004 (ubuntu-20.04-cloud)							
qemu	8005 (ubuntu-22.04-cloud)							
qemu	10044 (TID-PoT)	0.0 %	7.3 %	1.3% of 4 ...	14 days 02:1...	5.1% of 1C...		privateer
qemu	10045 (INESCTEC-CTI-1-GUI)	0.0 %	62.9 %	0.9% of 2 ...	14 days 21:5...	1.7% of 1C...		privateer
qemu	10057 (INESCTEC-CTI-2-GUI)	0.0 %	61.7 %	0.9% of 2 ...	12 days 11:4...	1.8% of 1C...		privateer
qemu	10060 (UCM-LoT)	0.0 %	85.3 %	1.7% of 4 ...	14 days 14:5...	6.9% of 1C...		privateer

Figure 4 - PRIVATEER components deployed on the integration environment

5 Workflow 1: Enhanced Security for Virtualized Infrastructures & Level of Trust Assessment

5.1 Attack models

Due to the complexity of the B5G infrastructure, several attacks might be feasible. When it comes to attacks against the virtualised infrastructure, the following categories are identified in [5]:

- **Software attacks:** Attacks targeting the software and firmware on the host system, including the operating system, hypervisor, BIOS, various software stacks, and any associated workloads.
- **Protocol attacks:** Attacks on protocols related to attestation and transporting workloads and data. These could compromise the integrity of a Trusted Execution Environment (TEE) instance, potentially leading to a breach of the workload or data.
- **Cryptographic attacks:** Vulnerabilities in cryptographic ciphers and algorithms can emerge over time due to advances in mathematics, computing power, or new computing paradigms like quantum computing. Maintaining crypto agility is crucial to replace outdated cryptographic methods with more secure alternatives. This flexibility is more feasible in software and firmware but generally impractical in hardware.
- **Basic physical attacks:** While extensive, intrusive attacks on CPUs are typically out-of-scope, other physical attacks such as cold DRAM extraction, bus and cache monitoring, or connecting attack devices to ports (e.g., PCIe, Firewire, USB-C) are considered within scope.
- **Basic upstream supply-chain attacks:** While attacks in the supply chain on Trusted Execution Environment (TEE) components are mostly out-of-scope, significant modifications such as adding debugging ports are considered within scope.

5.2 Workflow 1 implementation

One of the scopes of PRIVATEER is to protect not only data in rest and data in transit but also **data in use**, which is the latently introduced challenge. To perform this task

TEEs are employed and more specifically the Intel Software Guard Extensions (Intel SGX). SGX is a set of instructions designed to enhance the security of application code and data, by creating "SGX Enclaves" [6] Within these Enclaves sensitive code and data can be executed within an isolated, CPU-protected region. Through this application-level isolation, SGX manages to separate the enclave from the OS and system components (i.e., including hypervisor, BIOS, firmware and drivers). Hence, the aforementioned components, which are located outside the enclave, cannot access or modify the enclave and its data. Due to the isolation, the enclave cannot be accessed through traditional function calls, while data within enclaves can only be accessed by code that is part of the same enclave. It shall be noted that the enclave is isolated and protected through logically separating the memory of the trusted world. Such Root of Trust (RoT) is needed as an anchor, to provide the verifiable (trustworthiness) evidence monitoring based on which trust assessment is conducted. The PRIVATEER attestation framework is elaborated in D5.1.

Workflow #1 aims to demonstrate the use of the runtime attestation appraisals acquired leveraging the installed Security Probes (i.e., at the infrastructure) and μ Probes (i.e., within the containerized services) by the LoT Assessment component and the SCHEMA Privacy-aware Orchestrator for their decision-making.

The following table (i.e., Table 1) describes the functionalities implemented within the Release A of the PRIVATEER platform. Additionally, the functionalities that will be demonstrated in the Release B of the platform are described.

Table 1 - Workflow #1 components and functionalities

PRIVATEER Component	Implemented Functionalities	Functionalities for Release B
μ Probe	<ul style="list-style-type: none"> • Attestation Agents for producing verifiable evidence on static properties of the containerised application integrity (i.e., verifiable launch of the container) • Key Restriction Usage Policies 	<ul style="list-style-type: none"> • Integration of tracing capabilities (i.e., eBPF) for expanding the evidence from static properties to runtime properties, to monitor the configuration integrity of the containerised application • Verifiable Key Restriction Usage Policies
Security Probe	<ul style="list-style-type: none"> • Attestation Agents for producing verifiable evidence on runtime properties of the infrastructure configuration integrity • Verify μProbe signature 	<ul style="list-style-type: none"> • Extended tracing capabilities (i.e., eBPF) for control flow integrity • Attestation Agents for producing verifiable evidence on static properties of the container integrity (i.e., verifiable launch of the container)



<p>LoT Assessment Manager</p>	<ul style="list-style-type: none"> • Design & implementation of the first release of LoT assessment algorithm • Acquire and verify Attestation evidence for its evaluation by the LoT assessment component (time driven) • Acquire and verify Proof of Transit evidence for its evaluation by the LoT assessment component (event driven) • Generation of ‘New LoT value’ messages to be consumed by the Privacy-aware Orchestrator that will consider the new values in its orchestration decisions 	<ul style="list-style-type: none"> • Acquire and verify SLA accomplishment evidence for its evaluation by the LoT assessment component (time driven) • Acquire and verify CTI evidence for its evaluation by the LoT assessment component (event driven) • Acquire and verify Reputation evidence for its evaluation by the LoT assessment component (time driven) • Acquire and verify Privacy Index evidence for its evaluation by the LoT assessment component (time driven)
<p>SCHEMA Privacy-aware Orchestrator</p>	<ul style="list-style-type: none"> • Deployment of a simple Proof-of-Concept (PoC) evaluation-simulation scenario consisting of distinctive networking nodes (for the preliminary Privacy aware containerised application migration). • Digestion of the ‘New LoT value’ messages inside the core decision-engine of the Privacy aware Orchestrator. • Acquired technological possibility to export Network Service Graphs (post decision-making) from the SCHEMA Privacy-aware Orchestrator. • Implementation progress in the Game-theoretic procedures to encompass the ‘New LoT value’ messages. 	<ul style="list-style-type: none"> • Design and implement faster converging and more performance-efficient GT algorithms and Decentralized-AI policies (coded stochastic ADMM with sublinear convergence, Deep Q-Networks (DQN), clipped double-Q DDPG). • Adapt preliminary XAI features from the SCHEMA decision output (Machine Reasoning). • Solid evaluation of the VNF/container migration performance in terms of service latency, and Privacy leakage.
<p>Security Context Broker (SCB)</p>	<ul style="list-style-type: none"> • Acquire and verify attestation evidence Security Probe(s) • Acquire integrity appraisals from μProbe(s), through the Security Probe(s) • Acquire Trust Policies from LoT Assessment Manager 	<ul style="list-style-type: none"> • The acquiring of the attestation appraisals as well as the verification of signatures will be performed as part of the Secure Oracle (instead of the SCB) • Identity verification based on Privateer Self Sovereign Identities (SSI) technology and



	<ul style="list-style-type: none"> Acquire infrastructure information (i.e., container IDs and relationships/graph) for the deployed service from the SCHEMA Privacy-aware Orchestrator 	the use of Decentralized Identifiers (DIDs).
Privateer DLT (i.e., BESU)	<ul style="list-style-type: none"> Deployment and maintenance of all smart contracts for managing service trust requirements 	<ul style="list-style-type: none"> Attributes will be added into the smart contracts to enable Attribute-Based Access Control (ABAC), further leveraging the Verifiable Presentations (VPs) Implementation of the Trust Exposure Layer, providing harmonisation of information that led to the trust assessment
Secure Oracle	N/A	<ul style="list-style-type: none"> Acquire Attestation evidence verification enactment in the trusted environment of Secure Oracle (ported from SCB) Acquire Integrity appraisals from μProbe(s), Trust Policies from LoT Assessment Manager and Implementation of Attribute-Based Encryption (ABE) mechanisms to protect the access to the off-chain

5.2.1 Short Description

Beyond providing advanced protection through technologies like Intel SGX, PRIVATEER delivers the necessary trustworthiness appraisals for assessing trust levels during runtime. This is achieved by extracting attestation evidence from the TEE-enabled infrastructure. Attestation is the verification process that ensures the integrity and authenticity of the code running within an enclave. It validates the integrity of the enclave and confirms that it has not been altered and is operating the intended software. The extracted attestation evidence is made available to the Level of Trust (LoT) Assessment Manager and the SCHEMA Privacy-aware Orchestrator via the PRIVATEER Blockchain infrastructure, as detailed in D5.1 [4]. Blockchain technology offers a decentralized and tamper-proof method for storing and sharing evidence, guaranteeing its integrity and accessibility. PRIVATEER platform employs continuous monitoring and assessment of the enclave's configuration state. Hence, in case of unauthorized tampering, the attestation will fail, and the LoT Assessment Manager will promptly update its evaluation to reflect this failure, while the SCHEMA Privacy-



aware Orchestrator will also leverage this information for its decision-making (i.e., migrate a service or not).

Furthermore, it shall be noted that evidence is anticipated to be extracted from both the containerized applications (i.e., enclaves) and the infrastructure including the containers. In the initial phase of the PRIVATEER platform, the focus is on acquiring and assessing the attestation evidence stemming from the containerized applications. This is translated into a Level of Assurance (LoA) 2, as defined by ETSI [7].

As it pertains to the PRIVATEER designs to support the attestation of the configuration of the virtualised infrastructure, to initiate the collection of the evidence from the containerized application, certain steps are required. To begin with, a Trust Policy dedicated to the properties needed for the specific service (and its corresponding service graph chain) is needed by the Orchestrator in order to initiate the process within its virtualized infrastructure. This Trust Policy is made available to the Orchestrator through the Blockchain. Upon accessing it, the Orchestrator may send a Request for Evidence to the Security Probe installed at the server. The Security Probe will forward this request to the μ Probe installed at a container-level to initiate the collection of the evidence for the containerized application. The μ Probe will respond to the Security Probe with the collected evidence, and the latter will verify this information and forward it to the Blockchain. The following section will delve into the specific interactions and steps to enable this information exchange.

5.2.2 Detailed Description

5.2.2.1 Deployment of the infrastructure and containerised applications including Security Probes and μ Probes

Prior to the runtime collection of configuration integrity-related information, the steps referring to the deployment of the infrastructure should be described. This interaction reflects the secure deployment and launching of the containerised service (to-be-monitored; thus, attested) on the designated infrastructure. This further includes the setup of the Security Probes and μ Probes, which enable the configuration integrity verification.

The flow is initiated with a Service Provider (SP) requesting to deploy its service to a Mobile Network Operator's (MNO) infrastructure (see Figure 5). This request should include the application image and clearly specify the service, security, and privacy requirements from the SP's point of view (**step 1**). The SCHEMA Privacy-aware Orchestrator processes the request to construct the Interpretable Manifest (**step 2**). This manifest translates, in essence, the SP's requirements into the actual capabilities of the infrastructure. This manifest file further includes the gramine-enabled images, which are precomputed. This information is included in the Privacy Service Level Agreement (SLA). The containerized application is launched securely once the SP and MNO have both agreed to the terms outlined in the Privacy SLA.

This secure launching process includes two steps:

1. Deployment of the Container on the Infrastructure Side (Step 3): This includes the establishment of the virtualized infrastructure equipped with confidential computing capabilities (i.e., enacting upon SGX-capable nodes) for hosting the secure operation of the containerized applications as part of the deployed service-graph-chain. Core to this chain-of-trust is the verifiable launch of the necessary trust enablers (Security Probes) responsible for the monitoring and secure reporting of the configuration and behavioral evident based on which the trust assessment will be conducted - of both the infrastructure nodes as well as the host (containerized) microservices.
2. Deployment of the Containerized Application (Step 4): This involves the deployment of the containerized application, including the integration of μ Probes. These μ Probes are essential for monitoring and ensuring the integrity of the application/service.

As detailed in D5.1 [4], key restriction usage policies that enable the attestation of containers in a privacy-preserving manner are defined during these steps, for both Security Probes and μ Probes, by the SCHEMA Privacy-aware Orchestrator. More information on the JOIN phase for the Security Probes and μ Probes is available in D5.1 [4].

Upon the successful deployment of both the containerized applications and the infrastructure, a message signifying the verification of the enclave launch is sent from the container to the infrastructure (**step 5**). Similarly, the infrastructure sends a verification message back to the SCHEMA Privacy-aware Orchestrator (**step 6**). Consequently, the SCHEMA Privacy-aware Orchestrator is now aware that both the infrastructure and the containerized applications have been securely launched and can notify the Security Context Broker to create a new smart contract for the newly deployed service (**step 7**). Additionally, this notification from the SCHEMA Privacy-aware Orchestrator (**step 8**) enables the LoT (Level of Trust) Assessment to construct the Trust Policy for the newly deployed service (**step 9**). The Trust Policy is sent from the LoT Assessment component to the Security Context Broker (**step 10**) and the latter can construct and send to the Privateer DLT (i.e., BESU) the Smart Contract (SC), which contains the Service Graph Chain (SGC) Trust Chain Data Structure (**step 11**).

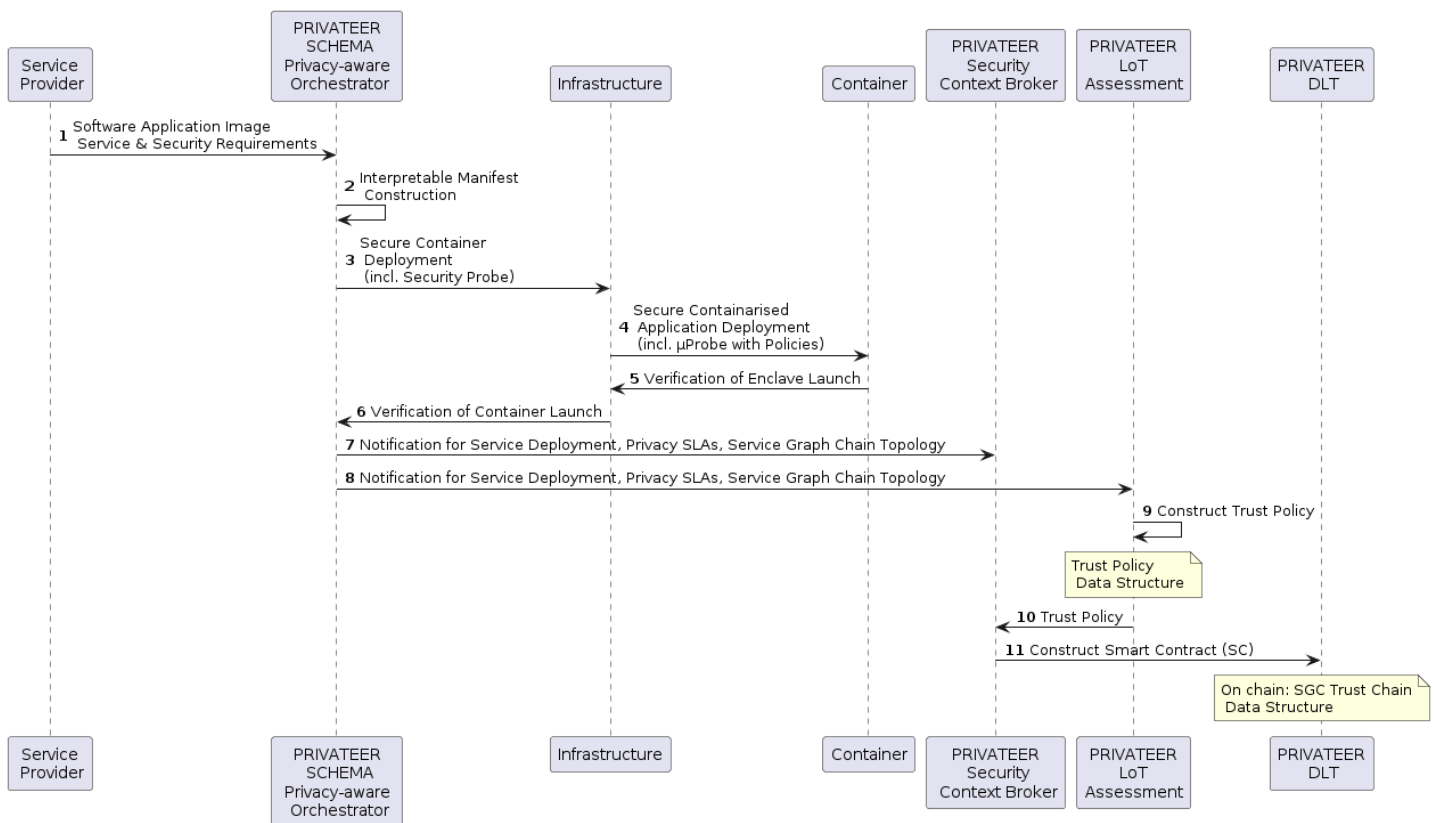


Figure 5 – Deployment of the infrastructure and containerised applications flow

5.2.2.2 Runtime LoT Assessment based on Attestation results from Security Probes and µProbes

After the infrastructure and the containerised services have been securely launched, the runtime configuration integrity verification operation may be initiated (see Figure 6). This involves the continuous attestation of both the infrastructure and the containerized applications by the Security Probes and µProbes respectively, to ensure that no tampering has occurred. The flow is initiated by the SCHEMA Privacy-aware Orchestrator, based on the Trust Policy constructed by the LoT Assessment component. The Security Context Broker (SCB) sends a request to the Security Probe to initiate the attestation flow (based on the predefined key restriction usage policies) **(step 1)**. Note that such a challenge can be sent periodically based on the Trust Policy for the specific service, as defined by the LoT Assessment component.

Upon receiving this request, the Security Probe will perform the configuration integrity verification (CIV) check for the infrastructure **(step 2)**, while it will further send and attestation initiation request to µProbes. The latter initiates the collection of the static properties to verify the integrity of the containerized application **(steps 3, 4)**. The response of the µProbe is returned to the Security Probe **(step 5)**, which verifies it **(step 6)**. Afterwards, the Security Probe constructs a JSON structure, named



attestation report, which encapsulates the aforementioned information, including the attestation appraisal of μ Probe, along with the Security Probe's evidence (**step 7**). This structure is sent by the Security Probe to the Security Context Broker (**step 8**). The latter, the Security Context Broker, is tasked with verifying the evidence of the Security Probe (**step 9**). Please note that in Release B this verification will be performed by the Secure Oracle.

After the successful verification of the received information, the Security Context Broker (SCB), builds the Trustworthiness Evidence Object Data Structure (**step 10**) and signs it, using its own key (**step 11**). The SCB further updates the smart contract, to include the Trustworthiness Evidence Object Data Structure and makes it available to other entities through the Privateer DLT (i.e., Hyperledger BESU²) (**step 12**). Other entities, such as the LoT Assessment component or the SCHEMA Privacy-aware Orchestrator may access this information through the Privateer DLT and leverage it for their trust evaluation and decision making respectively.

To do so, the LoT Assessment component may query the SCB for trustworthiness evidence, and more specifically attestation evidence in this case, based on the Container IDs that comprise the specific service graph chain (**step 13**). This information (i.e., Container IDs) is already known to the LoT assessment component from the secure deployment of the infrastructure phase (see step 8 of Figure 5). The SCB, in its turn will query the PRIVATEER DLT to locate this information (**step 14**) and respond back to the LoT assessment component with the needed information, containing the container appraisals and infrastructure evidence, stemming from the μ Probe and Security Probe respectively (**step 15**). The LoT assessment component leverages the attestation report to perform its trust evaluation and consequently sends the Actual Trust Level (ATL) data structure to the SCB (**step 16**). The latter may update the Smart Contract on the DLT, adding the ATL related field to the SGC Trust Chain data structure (**step 18**). In parallel, the LoT assessment component shares with the SCHEMA Privacy-aware Orchestrator the trust estimation (i.e., ATL) (**step 17**), to facilitate its decision-making process (i.e., migrate or not migrate container) (**step 19**).

Technically, though, the SCHEMA Privacy-aware Orchestrator shall be capable to perform not only binary decision-making process (i.e., migrate or not migrate container), but also deploy "*horizontal scaling*" and "*vertical scaling*", respectively. Thus, the current implementation of SCHEMA is fully modular; its AI-agents are modular and can be configured with additional functionality (e.g., container smart HPA - Predictive Horizontal Pod Autoscaler (HPA)). The heart of SCHEMA decision-making is its bidding mechanism. Smart negotiation tactics are being used between the agents, such as voting or auction. Surplus lies underneath a fully distributed system that avoids any kind of central point of failure. Finally, there is the capability

² <https://www.hyperledger.org/projects/besu>



for automatic agent discovery on-demand, something that is quite useful for the meta-information export of SCHEMA (i.e., the Security Context Broker (SCB)).

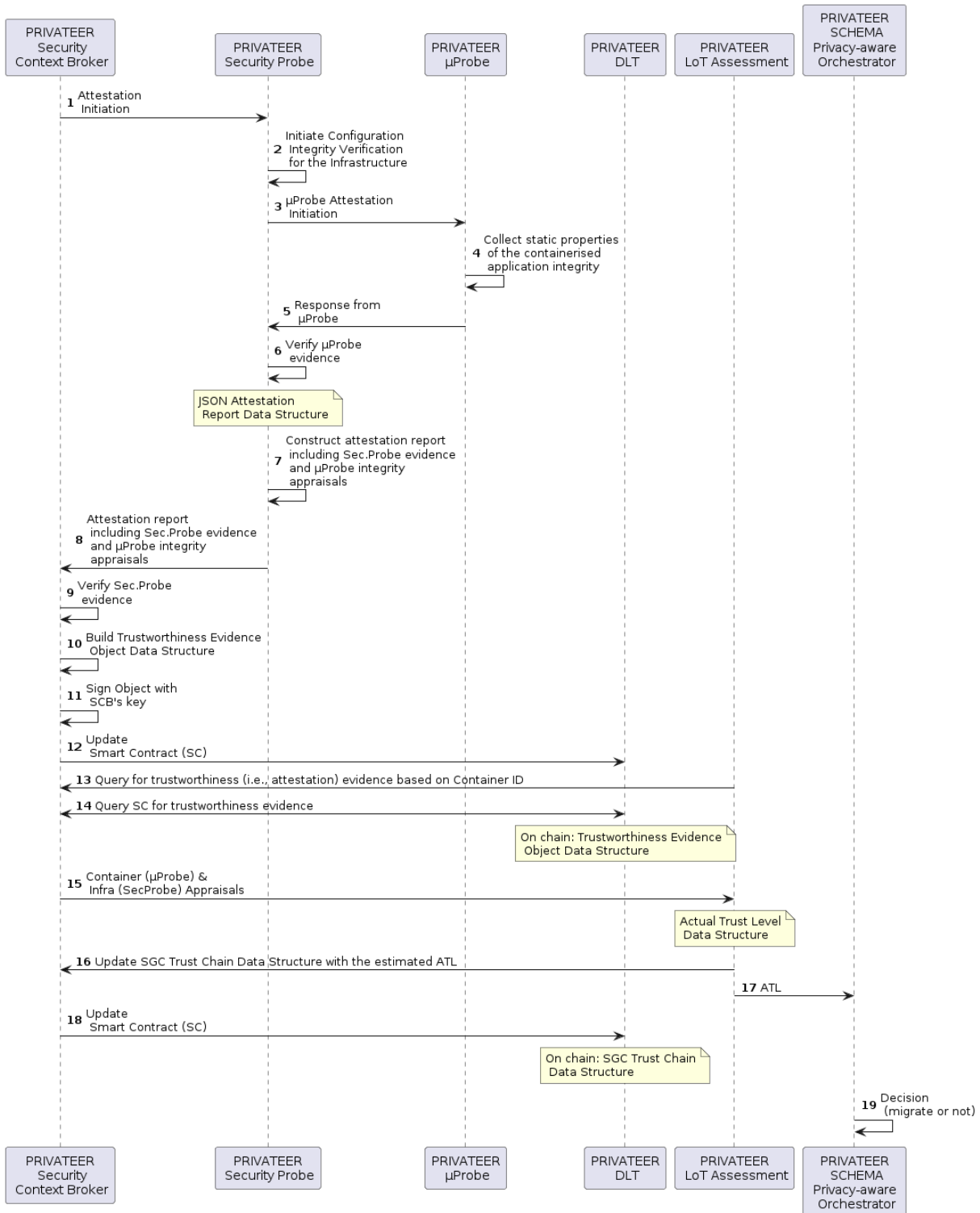


Figure 6 - Runtime LoT Assessment based on Attestation results from Security Probes and μProbes

5.2.3 APIs/interfaces

5.2.3.1 Step #1 API-ID: INT_SCB_DLT Storage and consumption of DLT Information.

Description: The Secure Context Broker (SCB) acts as a mediator for accessing the DLT, providing the necessary interfaces for storing and consuming DLT information. More specifically, the DLT is used to store and access trust-related information, including trust policies, trustworthiness evidence(s) and the trust value(s). The SCB constructs the smart contracts enabling the transactions within the DLT; thus, it can notify other entities (i.e., LoT Assessment Manager) when new information (i.e., attestation evidence) is received.

In Release B, certain functionalities of the SCB (i.e., acquiring attestation evidence from Security Probes as well as acquiring integrity appraisals μ Probes and Trust Policies from LoT Assessment Manager) will be performed by the Secure Oracle (as mentioned in Table 1).

Component: Secure Context Broker

Details: Prior to constructing the smart contracts for enabling the transactions on the DLT, the network should be initiated. Figure 7 illustrates the initiation of the Hyperledger BESU network, which is leveraged in PRIVATEER for storing information related to the trust assessment, including the trustworthiness evidence.



```

manos@manos-UBI: ~/quorum-test-network
[+] Running 28/28
✓ Network quorum-dev-quickstart Created 0.1s
✓ Volume "quorum-test-network_grafana" Created 0.0s
✓ Volume "quorum-test-network_prometheus" Created 0.0s
✓ Volume "quorum-test-network_blockscoutpostgres" Created 0.0s
✓ Container blockscoutpostgres Started 1.3s
✓ Container quorum-test-network-loki-1 Started 1.3s
✓ Container quorum-test-network-validator1-1 Started 1.1s
✓ Container quorum-test-network-promtail-1 Started 1.3s
✓ Container quorum-test-network-prometheus-1 Started 1.3s
✓ Container quorum-test-network-member2tessera-1 Started 1.3s
✓ Container quorum-test-network-member3tessera-1 Started 0.7s
✓ Container quorum-test-network-member1tessera-1 Started 1.1s
✓ Container chainlensmongodb Started 1.3s
✓ Container quorum-test-network-grafana-1 Started 1.3s
✓ Container blockscout Started 2.5s
✓ Container quorum-test-network-member1besu-1 Started 2.4s
✓ Container rpcnode Started 2.2s
✓ Container quorum-test-network-validator3-1 Started 2.2s
✓ Container quorum-test-network-member2besu-1 Started 2.4s
✓ Container quorum-test-network-validator2-1 Started 2.1s
✓ Container quorum-test-network-validator4-1 Started 2.1s
✓ Container quorum-test-network-member3besu-1 Started 2.2s
✓ Container chainlensapi Started 2.3s
✓ Container chainlensweb Started 4.2s
✓ Container chainlensingestion Started 4.2s
✓ Container quorum-test-network-explorer-1 Started 4.3s
✓ Container quorum-test-network-ethsignerProxy-1 Started 3.9s
✓ Container chainlensnglnx Started 4.9s
*****
Quorum Dev Quickstart
*****
-----
List endpoints and services
-----
JSON-RPC HTTP service endpoint : http://localhost:8545
JSON-RPC WebSocket service endpoint : ws://localhost:8546
Web block explorer address : http://localhost:25000/explorer/nodes
Chainlens address : http://localhost:8081/
Blockscout address : http://localhost:26000/
Prometheus address : http://localhost:9090/graph
Grafana address : http://localhost:3000/d/XE4V0NGZz/besu-overview?orgId=1&refresh=10s&from=now-30m&to=now&var-system=All
Collated logs using Grafana and Loki : http://localhost:3000/d/Ak6eXLSPxFemKYKEXfCH/quorum-logs-loki?orgId=1&var-ap
p=besu&var-search=

For more information on the endpoints and services, refer to README.md in the installation directory.
*****
manos@manos-UBI:~/quorum-test-network$

```

Figure 7 – Hyperledger BESU network Initiation

The Security Context Broker may support the functionalities of posting request or getting request from the Hyperledger BESU. The following figure (i.e., Figure 8) presents how the SCB sends (i.e., setTrustSource) and receives information (i.e., getTrustSource) to and from the Hyperledger BESU network. The SGC Smart Contract structure, available on-chain and accessible through the Hyperledger BESU network is reported in D5.1.

```

manos@manos-UBI: ~/quorum-test-network/app
manos@manos-UBI:~/quorum-test-network/app$ curl --location --request POST 'http://localhost:3001/api/setTrustSource' --header 'Content-Type: application/json' --data-raw '{
  "dbPointer": "example dbpointer",
  "name": "Example Name",
  "containerId": "example_containerId",
  "containerType": 1,
  "containerAppraisal": 2,
  "infrastructureAppraisal": 3,
  "oracleSignature": "example signature"
}'
{"transactionHash": "0xc6513dbccc7ce73a6027388f8a303a79d6d6a7e892bd7c0b85e96d9a910b9570", "blockHash": "0x198f72be69778f6a0c326224f7a89bdfbf89dd69037cc85ab3a13d81f255c37", "blockNumber": "65919", "gasUsed": "110623"}
manos@manos-UBI:~/quorum-test-network/app$ curl --location --request POST 'http://localhost:3001/api/getTrustSource/0x627306090abaB3A6e1400e9345bC60c78a8BEf57'
{"dbPointer": "example dbpointer", "name": "Example Name", "containerId": "example_containerId", "containerType": "1", "containerAppraisal": "2", "infrastructureAppraisal": "3", "oracleSignature": "example signature"}
manos@manos-UBI:~/quorum-test-network/app$

```

Figure 8 - SCB's post request and get request to Hyperledger BESU

5.2.3.2 Step #2 API-ID: INT_ORC Service Deployment to the underlying infrastructure

Description: The Privacy-aware Orchestrator is the entity responsible for the decision-making of a service deployment on a given infrastructure, considering the service and security requirements as well as the agreed SLA. Besides networking SLAs, inside the scope of PRIVATEER, we endorse the incorporation of Cybersecurity-related SSLAs, that grant Security & Privacy enablers for the UCs (like LoT assessment). The manner the SCHEMA (Privacy Aware Orchestrator) digests the messages is through a Trustworthiness Evidence Object Data Structure that is published via the Kafka bus and is consumed by the LoT service. As mentioned earlier, the attestation frequency is being inferred into the SCHEMA via the Kafka bus channel broker.

Component: Privacy Aware Orchestrator

Details: Since the SCHEMA orchestrator works with internal confidence levels (bidding mechanism), together with solving a multi-constraint game-theoretic optimization problem (e.g., where to service migrate with Privacy best-effort), the demonstrated workflow interacts in a few crucial steps with the rest I/O components.

The LoT publishes a message to the Kafka bus with the new LoT metric for such a service and the Orchestrator consumes it, at first. Such confidence levels appear critical not only for the Attestation engine itself, but also for the manner the (OpenAI Gym) game that empowers the decision-engine of SCHEMA will converge and, of course, which decision will take (migrate or not the particular service). The Attestation service provides a Trustworthiness Evidence value that, once processed by the LoT assessment service, results in a lower LoT, actually a LoT value below this service's LoT

tailored to each deployed service and dictate what information is needed (i.e., attestation evidence) along with other information such as periodicity.

Component: Privacy Aware Orchestrator and LoT Assessment Manager

Details: Here, the Privacy Aware Orchestrator shares the graph with the SCB. The LoT Assessment Manager receives this information to create the Trust Policies, as described in step 4.

5.2.3.4 Step #4 API-ID: INT_LOT_SCB Trust policy specification.

Description: This step (linked to step #3 and #1) allows the LoT Assessment Manager to share the Trust Policies to the DLT, through the SCB.

In Release A, this communication between the LoT Assessment Manager and the SCB is facilitated by Kafka.

Component: LoT Assessment Manager

Details: Figure 10 demonstrates the JSON structure as it pertains to the Trust Policy, as defined by the LoT Assessment Manager. Trust Policies are defined per service (i.e., based on the serviceID), along with the required Trust Level for the specific service, the Trust Property and the Trust Model. In this particular case, examined for the purposes of Workflow #1, the trust property under investigation is integrity. In other cases, different Trust Properties such as privacy or safety could be examined. Moreover, the Trust Model captures the trust relationships between the nodes that participate in the service graph chain, while the list of Trust Sources defines the specific dimensions where information for the LoT Assessment is required (i.e., SLA adherence, Privacy Index, Attestation, PoT, and CTI). In Workflow #1 the focus will be placed on Attestation. For each of the dimensions within the Trust Sources, periodicity, serviceID and trustModelID is defined.

```
{
  "requiredTrustLevel": "High", // Very Low, Low, Medium, High (Target Integrity Level
for Specific property)
  "trustProperty": "integrity",
  "serviceId": "5dda1e5a-8408-44d1-90ea-eddfc6e2a762",
  "trustModel": [
    {
      "trustModelId": "603ccedc-d476-4a25-b04e-8bc1446a1fd5",
      "edges": [ ] // {"src": "source-container-ID", "dst": "destination-container-ID"}
    }
  ],
  "listOfTrustSources": [
    {
      "name": "Attestation",
      "periodicity": 5,
      "serviceId": "5dda1e5a-8408-44d1-90ea-eddfc6e2a762",
      "targetTrustModelIds": [ "603ccedc-d476-4a25-b04e-8bc1446a1fd5" ]
    }
  ],
  "signature": "asd123dq122321123cdec" // LoT signature over the rest of the structure
}
```

Figure 10 - Trust Policy JSON structure



5.2.3.5 Step #5 API-ID: INT_SCB_SP Request for attestation.

Description: Based on the Trust Policy (acquired by the SCB from the LoT Assessment Manager in step #4), the SCB may initiate the attestation process by sending a nonce to the Security probes that are part of the infrastructure nodes hosting the deployed service graph chain, with a pre-defined (by the Trust Policy) periodicity.

In Release A, this communication between the SCB and the Security Probe(s) is facilitated by Kafka.

Component: SCB

Details: The SCB is responsible for processing the Trust Policy (as defined in Figure 10) and then creating the nonce to be sent to the Security Probe and the μ Probe. Hence this step includes the creation of the nonce which is shared with the Security Probe.

5.2.3.6 Step #6 API-ID: INT_SP_SCB Attestation report

Description: Following the attestation request (step #5), the Security Probe(s) that comprise the service graph chain (for a given service), report back to the SCB with a structure that includes the signature of the nonce, along with other information such as the integrity appraisals for the μ Probe(s) and timestamps. This structure is made accessible to other entities by the SCB through the DLT (see step #1).

In Release A, this communication between the Security Probe(s) and SCB is facilitated by Kafka.

Component: Security Probe(s)

Details: Upon receiving the attestation request from the SCB, the Security Probe(s), that participate in the given service graph chain, initiate the Configuration Integrity Verification (CIV) flow, elaborated in D5.1. To perform this task, the infrastructure node needs to have been instantiated with a CoCo which in our case is Gramine. Figure 11 demonstrates Gramine's deployment in the Intel SGX box.



```

Gramine is starting. Parsing TOML manifest file, this may take some time...
(host_main.c:967:load_enclave) debug: Gramine parsed TOML manifest file successfully
(host_framework.c:225:create_enclave) debug: Enclave created:
(host_framework.c:226:create_enclave) debug:   base:           0x0000000000000000
(host_framework.c:227:create_enclave) debug:   size:            0x0000000010000000
(host_framework.c:228:create_enclave) debug:   misc_select:    0x00000000
(host_framework.c:229:create_enclave) debug:   attr.flags:     0x0000000000000005
(host_framework.c:230:create_enclave) debug:   attr.xfrm:      0x000000000000000e7
(host_framework.c:231:create_enclave) debug:   ssa_frame_size: 4
(host_framework.c:232:create_enclave) debug:   isv_prod_id:   0x00000000
(host_framework.c:233:create_enclave) debug:   isv_svn:       0x00000000
(host_main.c:480:initialize_enclave) debug: Adding pages to SGX enclave, this may take some time...
(host_framework.c:313:add_pages_to_enclave) debug: Adding pages to enclave: 0xfd4f000-0x10000000 [REG:R--] (manifest) measured
(host_framework.c:313:add_pages_to_enclave) debug: Adding pages to enclave: 0xfd2f000-0xfd4f000 [REG:RW-] (ssa) measured
(host_framework.c:313:add_pages_to_enclave) debug: Adding pages to enclave: 0xfd2b000-0xfd2f000 [TCS:---] (tcs) measured
(host_framework.c:313:add_pages_to_enclave) debug: Adding pages to enclave: 0xfd27000-0xfd2b000 [REG:RW-] (tls) measured
(host_framework.c:313:add_pages_to_enclave) debug: Adding pages to enclave: 0xfce7000-0xfd27000 [REG:RW-] (stack) measured
(host_framework.c:313:add_pages_to_enclave) debug: Adding pages to enclave: 0xfca7000-0xfce7000 [REG:RW-] (stack) measured
(host_framework.c:313:add_pages_to_enclave) debug: Adding pages to enclave: 0xfc67000-0xfca7000 [REG:RW-] (stack) measured
(host_framework.c:313:add_pages_to_enclave) debug: Adding pages to enclave: 0xfc27000-0xfc67000 [REG:RW-] (stack) measured
(host_framework.c:313:add_pages_to_enclave) debug: Adding pages to enclave: 0xfc17000-0xfc27000 [REG:RW-] (sig_stack) measured
(host_framework.c:313:add_pages_to_enclave) debug: Adding pages to enclave: 0xfc07000-0xfc17000 [REG:RW-] (sig_stack) measured
(host_framework.c:313:add_pages_to_enclave) debug: Adding pages to enclave: 0xfbf7000-0xfc07000 [REG:RW-] (sig_stack) measured
(host_framework.c:313:add_pages_to_enclave) debug: Adding pages to enclave: 0xfbe7000-0xfbf7000 [REG:RW-] (sig_stack) measured
(host_framework.c:313:add_pages_to_enclave) debug: Adding pages to enclave: 0xfb8d000-0xfbdd000 [REG:RW-X] (code) measured
(host_framework.c:313:add_pages_to_enclave) debug: Adding pages to enclave: 0xfbdd000-0xfbe7000 [REG:RW-] (data) measured
(host_framework.c:313:add_pages_to_enclave) debug: Adding pages to enclave: 0xfbe2000-0xfbd000 [REG:RW-] (bss) measured
(host_framework.c:313:add_pages_to_enclave) debug: Adding pages to enclave: 0x10000-0xfbd000 [REG:RWX] (free)
(host_main.c:562:initialize_enclave) debug: Added all pages to SGX enclave
(host_framework.c:516:init_enclave) debug: Enclave initializing:
(host_framework.c:517:init_enclave) debug:   enclave id: 0x00000000ffff000
(host_framework.c:518:init_enclave) debug:   mr_enclave: 30d708d7601cd1d117f3ac7fd43bb922c41f009ca27b302c2bc9d63b7fbb0b6
-----
Gramine detected the following insecure configurations:
- loader.log_level = warning|debug|trace|all (verbose log level, may leak information)
- loader.insecure_use_cmdline_argv = true (forwarding command-line args from untrusted host to the app)
- sgx.allowed_files = [ ... ] (some files are passed through from untrusted host without verification)
Gramine will continue application execution, but this configuration must not be used in production!

```

Figure 11 - Gramine deployment on Intel SGX

When the infrastructure is properly configured, the Configuration Integrity Verification (CIV) process may begin for both the Security Probes and the μ Probes. This includes: i) the creation of the attestation keys, ii) the correct execution of a predefined Key Restriction Usage Policy, iii) the successful sign of the policy, iv) the policy authorise correct execution and v) the successful creation of the attestation evidence. The aforementioned steps verify, in essence that the attestation evidence is strictly signed when the Security Probe (or the μ Probe) adhere to the enforced policy. The logs of this process are illustrated in Figure 12.

```

ubuntu@ubuntu:~/Connect/SGX_CIV$ python3 test.py
[CIV LIB] SIGNATURE CREATED SUCCESSFULLY
[CIV LIB] SIGNATURE CREATED SUCCESSFULLY
{"signature": "3945622078248c1496592fa830b0d95afbcd553ae34a94ff6d09a8eF661b2c25bF327fa202210ef983f4eab01b95824d981535e9fd09aeb46e72c675988c78644da503bd4d8800", "signedDigest": "91a48ff8a
f89dcd039fd62aa7e95f89158d586ff43931e6b94e7dc128fe9d3", "attestation_PubKey": "-----BEGIN PUBLIC KEY-----\nMFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEHs5TRVesVx8XkMh0FAF5V1ecknFlnNGtUG0Lmk5G
qM7R/zXyENS+HECCAL1t6I0150xPpJLdx5XmFhGazQ==\n-----END PUBLIC KEY-----\n"}
ubuntu@ubuntu:~/Connect/SGX_CIV$ tail -f debug.out
(host_framework.c:313:add_pages_to_enclave) debug: Adding pages to enclave: 0x2bd0800-0x2bd2000 [REG:RW-] (data) measured

```

Figure 12 - Configuration Integrity Verification logs

The response sent back from the Security Probe to the SCB is illustrated in Figure 13, including the result (i.e., appraisal), set either to 0 or 1 to indicate failure or success respectively for both the Security Probe and the μ Probe, along with other information such as the timestamp, nonce and the signature.

```

{ "attestationReports": [
  {
    "containerID": "Microservice 1  $\mu$ Probe",
    "attestationReport": [
      {
        "claim": "container-configuration-integrity",
        "timestamp": "2024-05-16T15:30:45Z",
        "appraisal": 1
      }, {
        "claim": "runtime-integrity",
        "timestamp": "2024-05-16T15:35:22Z",
        "appraisal": 0
      }
    ]
  }
]

```

```

    }
  ]
},
"securityProbeEvidence": {
  "timestamp": "2024-05-16T15:30:45Z",
  "nonce": "d78080092edf3633e6933f67ddfe6744",
  "signatureAlgorithmType": "ECDSA-SHA256",
  "signature": "30440220655e8f8b6f96a6...",
  "keyRef": "ecdsa_public_key_71"
}}

```

Figure 13 - Attestation report in JSON

5.2.3.7 Step #7 API-ID: INT_LOT_SCB Consume JSON structure from Smart Contract and Report LoT new value

Description: This step (linked to step #6) enables the LoT Assessment Manager to consume information regarding the attestation as reported to the DLT by the Security Probe through the SCB. This type of information allows the LoT Assessment Manager to calculate the LoT value.

In Release A, this communication between the SCB and the LoT Assessment Manager is facilitated by Kafka.

Upon calculating the LoT value (step #8) the LoT Assessment Manager reports this value to the DLT through the SCB.

Component: LoT

Details:

Reading and Processing Messages: Messages produced by Attestation (Figure 14) are read after we include them as a part of the LoT evaluation, and the input is processed.

```

dbPointer": "f91202b5-4359-49da-8ab4-ea7bffc7b52e", "categoryTrustSource": [{"name": "Attestation", "containerId": "223f9264-2820-4d1b-9f6e-921415906105", "containerType": "0", "containerAppraisal": "1", "infrastructureAppraisal": "1", "timestamp": "2023-12-12T"}], {"name": "PoT", "trustModelId": "223f9264-2820-4d1b-9f6e-921415906105", "potScore": "1", "infrastructureAppraisal": "1", "timestamp": "2023-12-12T"}], "oracleSignature": "ecc3327a94893390841cb9ae217d4d49377709e77d79c356ea7b585228cde1c2f8327097ef6591bd8f88a2ba25fcf7b9c2f6c221c1d7d5298b4f758cd2ec93d"}

```

Figure 14 - Attestation message received

Task Creation: A new task is created based on the processed messages (15) and inserted into a new row in the MongoDB Database (Figure 16).

```

2024-07-05T15:07:33.941Z INFO 1 --- [data-collector] [ntainer#2-0-C-1] o.u.d.s.impl.AggregationServiceImpl : ATTESTATION_DATA:: {categoryTrustSource=[{name=Attestation, containerId=223f9264-2820-4d1b-9f6e-921415906105, containerType=0, containerAppraisal=1, infrastructureAppraisal=1, timestamp=2023-12-12T}], dbPointer=f91202b5-4359-49da-8ab4-ea7bffc7b52e, infrastructureAppraisal=1, timestamp=2023-12-12T}, {name=PoT, trustModelId=223f9264-2820-4d1b-9f6e-921415906105, potScore=1, infrastructureAppraisal=1, timestamp=2023-12-12T}], oracleSignature=ecc3327a94893390841cb9ae217d4d49377709e77d79c356ea7b585228cde1c2f8327097ef6591bd8f88a2ba25fcf7b9c2f6c221c1d7d5298b4f758cd2ec93d}
2024-07-05T15:07:33.942Z INFO 1 --- [data-collector] [ntainer#2-0-C-1] o.u.d.s.impl.AggregationServiceImpl : TaskInfo attestation -> org.ucm.datacollector.model.TaskInfo@442b2038

```

Figure 15 - Task created

```

_id: ObjectId('66880c359cf44f3aaeb67560')
driven_type: "WARN"
created_by: "ATTESTATION"
event_date: 2024-07-05T15:07:33.942+00:00
task_date: 2024-07-05T15:07:33.942+00:00
status: 1
_class: "org.ucm.datacollector.model.lot.entity.Task"
    
```

Figure 16 - Task Table

Fuzzy Logic Evaluation (Step 1): The first step of fuzzy logic is evaluating the inputs that we receive based on Time-driven or Event-driven logic (Figure 17), which is saved on DB (Figure 18).

```

{"id":"66880c35ff5665075ec10b7c","taskId":"66880c359cf44f3aaeb67560","recordDate":[2024,7,5,15,7,33,976567083],"d1":49.32653668745014,"d2":50.00000000000004,"d3":50.00000000000056,"d4":83.299999999999923,"d5":83.299999999999923,"d6":null}
    
```

Figure 17 - Result of the first step of LoT Assessment

```

_id: ObjectId('66880c35ff5665075ec10b7c')
task_id: "66880c359cf44f3aaeb67560"
D1: 49.32653668745014
D2: 50.00000000000004
D3: 50.000000000000056
D4: 83.299999999999923
D5: 83.299999999999923
record_date: 2024-07-05T15:07:33.976+00:00
_class: "org.ucm.lotfuzzy.model.lot.entity.DimensionsAssessment"
    
```

Figure 18 - Dimensions_Result Table

LoT Evaluation: The LoT evaluation is completed, providing an overall assessment (Figure 19) and saved on DB (Figure 20).

```

{"recordDate":"2024-07-05T15:07:34.007583844","trustLevel":"61.05243403690463","serviceId":"dcb3627-22a5-4d90-9d30-ee14782db57f"}
    
```

Figure 19 Final LoT Evaluation Result shown by Kafka message

```

_id: ObjectId('66880c36ff5665075ec10b7d')
trust_level: 61.05243403690463
task_id: "66880c359cf44f3aaeb67560"
record_date: 2024-07-05T15:07:34.007+00:00
_class: "org.ucm.lotfuzzy.model.lot.entity.FinalTrust"
    
```

Figure 20 Final_Trust Table

The JSON structure (Figure 21) captures the initial set of data required for the LoT evaluation process. It includes unique identifiers, precise timestamps, and a series of evaluated metrics, thereby providing a foundation for assessing the trustworthiness of a service within a distributed system. Handling these data through Kafka and Docker ensures scalability, reliability, and real-time processing capabilities, making them an integral part of a robust trust evaluation framework.

First Step of LoT Evaluation

In the context of Level of Trust (LoT) evaluation, the first step typically involves collecting and recording relevant metrics or dimensions that contribute to the overall trust assessment. A detailed explanation of this process is provided in the JSON:

- **Data Collection:**
In the first two lines, we created a unique task ID for executing the LoT assessment for every call. Using `service_id` and `task_id`, we can track the process until it is completed.
- **Timestamp Recording:**
Precise timestamps were recorded to ensure that the data were correlated with other events within the system. This is crucial for maintaining the integrity of the evaluation process and auditing purposes.
- **Dimension Evaluation:**
Each dimension was evaluated and assigned a numerical value, which was derived from the system's first-step assessments.
For instance, `d4` represents an attestation metric, `d5` represents a Proof of Transit metric, and so forth. The actual interpretation of these dimensions depends on the specific implementation of the LoT evaluation process (Fuzzy Logic).

This JSON structure is part of a broader workflow involving multiple services and components, such as Kafka for messaging and Docker for containerization. The data captured here will be published to a Kafka topic and DB, consumed by the Attestation Service, processed, and then fed back into the second step of the LoT Assessment for continuous trust assessment and management.

```

{
  "id": "66854654fc706d7389ba3834",
  "taskId": "6685464002828f5b0d3dfc06",
  "recordDate": [
    2024,
    7,
    3,
    12,
    38,
    44,
    342889564
  ],
  "d1": 37.2729011666442,
  "d2": 68.71254594071655,
  "d3": 46.02591373812544,
  "d4": 49.99760697655505,
  "d5": 83.29999999999923,
  "d6": null
}

```

Figure 21 - First Step Result JSON file

Figure 22 depicts a JSON object that represents the final trust result for a specific system. We break down each component of the JSON object and explain its significance.

- **recordDate:**
 Description: This field indicates the date and time at which the trust level is recorded.
 Format: The timestamp is in the ISO 8601 format, which includes the date, time, and fractional seconds.
- **trustLevel:**
 Description: This field represents the computed trust level for a particular service.
- **serviceId:**
 Description: This field is a unique identifier for the service being evaluated.
 Format: Identifier is a Universally Unique Identifier (UUID) that ensures a globally unique reference.

```

1  {
2    "recordDate": "2024-07-01T12:15:24.354203479",
3    "trustLevel": "42.85267762108336",
4    "serviceId": "S57310caf-bcc4-4008-82b8-6cfa6263bbcd"
5  }

```

Figure 22 - Final LOT assessment JSON file

The image illustrates the final trust result being sent to a Kafka broker and then saved for further use by different modules within a system.



5.2.3.8 Step #8 API-ID: INT_ORC_SCB Consume LoT value and Migration decision.

Description: Following step #7, the LoT value stored on the DLT can be accessed by the Privacy Aware Orchestrator, through the SCB. In Release A, this communication between Privacy Aware Orchestrator and the SCB is facilitated by Kafka.

Leveraging the LoT value acquired by the DLT through the SCB and calculated by the LoT Assessment Manager, the Privacy Aware Orchestrator may now use this metric for its own decision-making. Hence, if a service receives a low value, the Orchestrator may decide to migrate it.

Component: Privacy Aware Orchestrator

Details: Once the SCHEMA orchestrator digests the LoT Assessment messages from the LoT Assessment Manager through Kafka, it initiates its internal **innovative containerized applications migration** procedures (**service orchestrator functions**). SCHEMA is running with SFC Management with Distributed Reinforcement Learning and Game-Theory. In this specific workflow #1, we utilized the following hard-coded steps across a fully ad-hoc simulation environment:

- Proposed solution: We used a framework that attaches distributed Reinforcement Learning agents to perform intra-domain service migration & slicing orchestration locally and introduced an auction mechanism to allow agents to exchange container services.
- We separated the network into an intra-domain level graph and multiple intra-domain level graphs.
- We assigned local domain Reinforcement Local agents responsible for performing intra-domain containerized applications migration & orchestration.
- For every containerized service in the network, we performed an auction and thus letting multiple local domain RL agents bid to receive a specific container and migrate it to the selected server.
- The local RL agents performed intra-domain orchestration (*migrate, or not the service; based on the LoT messages prone*) to avoid global network SFC reconfiguration.



```

Entering the hyperloop.
CHECKPOINT REACHED --> WAITING_CP-A
Timeout threshold: 0% | 0/120 [00:00<?, ?it/s]
All 2 agents time-synced in checkpoint with status: WAITING_CP-A
CHECKPOINT COMPLETE. SYNC DONE
bfc893d77d2ea791700f48895cd36e74aa27a42c86056ebf5da100b26edbf3e: {'uid': 'bfc893d77d2ea791700f48895cd36e74aa27a42c86056ebf5da100b26edbf3e', 'name': 'confident_mendel', 'image': 'nginx:latest', 'hosted_uid': '4ba1fe20-d834-4cb2-9c3b-30f08d1743bf', 'hosted_ip': '10.55.175.94', 'cpu': 48763933, 'ram': 6815744, 'status': 'running'}
[{'uid': 'bfc893d77d2ea791700f48895cd36e74aa27a42c86056ebf5da100b26edbf3e', 'name': 'confident_mendel', 'image': 'nginx:latest', 'hosted_uid': '4ba1fe20-d834-4cb2-9c3b-30f08d1743bf', 'hosted_ip': '10.55.175.94', 'cpu': 48763933, 'ram': 6815744, 'status': 'running'}]
[{'uid': 'bfc893d77d2ea791700f48895cd36e74aa27a42c86056ebf5da100b26edbf3e'}]
cont in cont:
|-- bfc893d77d2ea791700f48895cd36e74aa27a42c86056ebf5da100b26edbf3e
CC bfc893d77d2ea791700f48895cd36e74aa27a42c86056ebf5da100b26edbf3e already registered, ckecking IP
Listing functions data downloaded:
bfc893d77d2ea791700f48895cd36e74aa27a42c86056ebf5da100b26edbf3e: {'uid': 'bfc893d77d2ea791700f48895cd36e74aa27a42c86056ebf5da100b26edbf3e', 'name': 'confident_mendel', 'image': 'nginx:latest', 'hosted_uid': '4ba1fe20-d834-4cb2-9c3b-30f08d1743bf', 'hosted_ip': '10.55.175.94', 'cpu': 48763933, 'ram': 6815744, 'status': 'running'}
[{'uid': 'bfc893d77d2ea791700f48895cd36e74aa27a42c86056ebf5da100b26edbf3e', 'name': 'confident_mendel', 'image': 'nginx:latest', 'hosted_uid': '4ba1fe20-d834-4cb2-9c3b-30f08d1743bf', 'hosted_ip': '10.55.175.94', 'cpu': 48763933, 'ram': 6815744, 'status': 'running'}]
0
li->0
lo->[{'uid': 'bfc893d77d2ea791700f48895cd36e74aa27a42c86056ebf5da100b26edbf3e', 'name': 'confident_mendel', 'image': 'nginx:latest', 'hosted_uid': '4ba1fe20-d834-4cb2-9c3b-30f08d1743bf', 'hosted_ip': '10.55.175.94', 'cpu': 48763933, 'ram': 6815744, 'status': 'running'}]
get_obs() result:
[48763933, 6815744, 0.5, 8.3, 83.8]
Agent bid is: 0
bfc893d77d2ea791700f48895cd36e74aa27a42c86056ebf5da100b26edbf3e: {'uid': 'bfc893d77d2ea791700f48895cd36e74aa27a42c86056ebf5da100b26edbf3e', 'name': 'confident_mendel', 'image': 'nginx:latest', 'hosted_uid': '4ba1fe20-d834-4cb2-9c3b-30f08d1743bf', 'hosted_ip': '10.55.175.94', 'cpu': 48763933, 'ram': 6815744, 'status': 'running'}
[{'uid': 'bfc893d77d2ea791700f48895cd36e74aa27a42c86056ebf5da100b26edbf3e', 'name': 'confident_mendel', 'image': 'nginx:latest', 'hosted_uid': '4ba1fe20-d834-4cb2-9c3b-30f08d1743bf', 'hosted_ip': '10.55.175.94', 'cpu': 48763933, 'ram': 6815744, 'status': 'running'}]
[{'uid': 'bfc893d77d2ea791700f48895cd36e74aa27a42c86056ebf5da100b26edbf3e'}]
cont in cont:
|-- bfc893d77d2ea791700f48895cd36e74aa27a42c86056ebf5da100b26edbf3e
CC bfc893d77d2ea791700f48895cd36e74aa27a42c86056ebf5da100b26edbf3e already registered, ckecking IP
AGENT_FUNCTION_INDEX: 0
CHECKPOINT REACHED --> WAITING_CP-B
Timeout threshold: 0% | 0/120 [00:00<?, ?it/s]
All 2 agents time-synced in checkpoint with status: WAITING_CP-B
CHECKPOINT COMPLETE. SYNC DONE
downloading the votes of other nodes...
nodes UUIDs and votes results:
[{'149d336f-c953-4257-92cf-f667050df16a', '4ba1fe20-d834-4cb2-9c3b-30f08d1743bf'}]
[0, 0]
winner UUID: 149d336f-c953-4257-92cf-f667050df16a Vote: 0
this node ISSOURCE
I AM THE SOURCE
docker image get: <image: nginx:latest>
Image saved to './outbox/transfer_image.tar' successfully.
Stopping/Killing container...
waiting 2 s...
done. moving on.
cont_id for removal: bfc893d77d2ea791700f48895cd36e74aa27a42c86056ebf5da100b26edbf3e
removed_container from DB OK: bfc893d77d2ea791700f48895cd36e74aa27a42c86056ebf5da100b26edbf3e
|-- CREATING LOCAL SSH CLIENT
|-- CONNECTING TO REMOTE HOST...
|-- CONNECTING VIA SFTP...
|-- UPLOADING...
|-- TERMINATING CONNECTION
-- CONTAINER TRANSFER COMPLETE!
-- CLEANING OUTBOX
[

```

Figure 9i - SCHEMA main deployment phase. We denote hereby a successful containerized application migration with the following depicted convention in the above figure: (a) green colour bar: Source <-> Destination containers for a specific LoT attested 5G/6G service, and (b) red colour bar: Indication message of successful service migration from SCHEMA Privacy-aware orchestrator (CLIENT INTERFACE SHELL).



```
(base) oki@SchemaNode4:~/mont6g-schema-iv/shell$ python3 main.py
MONT-6G BUSINESS INTERFACE SHELL
Initializing...
Loading configuration file
Configuration file loaded successfully
Successfully connected to Redis.

MONT-6G BUSINESS INTERFACE SHELL
Type "help" and press return for a command list
>>] nodes
Registered Nodes:
{'heartbeat': 1707063008,
 'ip': '10.55.175.94',
 'name': 'SchemaNode4',
 'opinion': '',
 'status': 'WAITING',
 'uid': '4ba1fe20-d834-4cb2-9c3b-30f08d1743bf'}
>>] nodes
Registered Nodes:
{'heartbeat': 1707063061,
 'ip': '10.55.175.89',
 'name': 'SchemaNode5',
 'opinion': '',
 'status': 'WAITING',
 'uid': '149d336f-c953-4257-92cf-f667050df16a'}
{'heartbeat': 1707063008,
 'ip': '10.55.175.94',
 'name': 'SchemaNode4',
 'opinion': '',
 'status': 'WAITING',
 'uid': '4ba1fe20-d834-4cb2-9c3b-30f08d1743bf'}
>>] start
Confirm System Startup (in 10s)? (y/n) >>] y
Updating DB...
Confirmed! Start time in 10s at: 2024-02-04 16:11:44
>>] █

I
```

Figure 9ii - SCHEMA main deployment phase (BUSINESS INTERFACE SHELL).



```

[{'uid': 'bfcd893d77d2ea791700f48895cd36e74aa27a42c86056ebf5da100b26edbf3e', 'name': 'confident_mendel', 'image': 'nginx:latest', 'hosted_uid': '4ba1fe20-d834-4cb2-9c3b-30f08d1743bf', 'hosted_ip': '10.55.175.94', 'cpu': 48763933, 'ram': 6815744, 'status': 'running'}]
li->0
lo->[{'uid': 'bfcd893d77d2ea791700f48895cd36e74aa27a42c86056ebf5da100b26edbf3e', 'name': 'confident_mendel', 'image': 'nginx:latest', 'hosted_uid': '4ba1fe20-d834-4cb2-9c3b-30f08d1743bf', 'hosted_ip': '10.55.175.94', 'cpu': 48763933, 'ram': 6815744, 'status': 'running'}]
get_obs() result:
[48763933, 6815744, 0.3, 7.6, 60.3]
Agent bid is: 0
bfcd893d77d2ea791700f48895cd36e74aa27a42c86056ebf5da100b26edbf3e: {'uid': 'bfcd893d77d2ea791700f48895cd36e74aa27a42c86056ebf5da100b26edbf3e', 'name': 'confident_mendel', 'image': 'nginx:latest', 'hosted_uid': '4ba1fe20-d834-4cb2-9c3b-30f08d1743bf', 'hosted_ip': '10.55.175.94', 'cpu': 48763933, 'ram': 6815744, 'status': 'running'}]
bfcd893d77d2ea791700f48895cd36e74aa27a42c86056ebf5da100b26edbf3e: {'uid': 'bfcd893d77d2ea791700f48895cd36e74aa27a42c86056ebf5da100b26edbf3e', 'name': 'confident_mendel', 'image': 'nginx:latest', 'hosted_uid': '4ba1fe20-d834-4cb2-9c3b-30f08d1743bf', 'hosted_ip': '10.55.175.94', 'cpu': 48763933, 'ram': 6815744, 'status': 'running'}]
cont in cont:
AGENT_FUNCTION_INDEX: 0
CHECKPOINT REACHED --> WAITING_CP-B
Timeout threshold: 0% | 0/120 [00:00<?, ?it/s]
All 2 agents time-synced in checkpoint with status: WAITING_CP-B
CHECKPOINT COMPLETE. SYNC DONE
downloading the votes of other nodes...
nodes UUIDs and votes results:
[{'149d336f-c953-4257-92cf-f667050df16a', '4ba1fe20-d834-4cb2-9c3b-30f08d1743bf'}]
[0, 0]
Winner UUID: 149d336f-c953-4257-92cf-f667050df16a Vote: 0
This node NOT SOURCE
WINNER - I AM THE DESTINATION
Waiting the container
---
File not found in the INBOX. Waiting for file...
[]
---
File not found in the INBOX. Waiting for file...
[]
---
File not found in the INBOX. Waiting for file...
[]
---
File not found in the INBOX. Waiting for file...
[]
---
File not found in the INBOX. Waiting for file...
[]
---
FILE FOUND! CONTINUE
F Exists? True
---
FOUND! Time guard 2s...
Container loaded OK!
saved container to db OK: {'uid': '67919c70a28735a2c7a3fcd4093b5dd17845f10f73535f29ecd3188e667c5ab6', 'name': 'blissful_satoshi', 'image': 'nginx:latest', 'hosted_uid': '149d336f-c953-4257-92cf-f667050df16a', 'hosted_ip': '10.55.175.89', 'cpu': 42036968, 'ram': 6787072, 'status': 'created'}
Transfer file removed successfully.
67919c70a28735a2c7a3fcd4093b5dd17845f10f73535f29ecd3188e667c5ab6: {'uid': '67919c70a28735a2c7a3fcd4093b5dd17845f10f73535f29ecd3188e667c5ab6', 'name': 'blissful_satoshi', 'image': 'nginx:latest', 'hosted_uid': '149d336f-c953-4257-92cf-f667050df16a', 'hosted_ip': '10.55.175.89', 'cpu': 42036968, 'ram': 6787072, 'status': 'created'}
67919c70a28735a2c7a3fcd4093b5dd17845f10f73535f29ecd3188e667c5ab6: {'uid': '67919c70a28735a2c7a3fcd4093b5dd17845f10f73535f29ecd3188e667c5ab6', 'name': 'blissful_satoshi', 'image': 'nginx:latest', 'hosted_uid': '149d336f-c953-4257-92cf-f667050df16a', 'hosted_ip': '10.55.175.89', 'cpu': 42036968, 'ram': 6787072, 'status': 'created'}]
[{'67919c70a28735a2c7a3fcd4093b5dd17845f10f73535f29ecd3188e667c5ab6'}]

```

Figure 9iii - SCHEMA main deployment phase. We denote hereby a successful containerized application migration with the following depicted convention in the above figure: (a) green colour bar: Source <-> Destination containers for a specific LoT attested 5G/6G service, (b) red colour bar: Indication message of successful service migration from SCHEMA Privacy-aware orchestrator, and (c) yellow colour bar: the SLAs/SSLAs constraints affecting the SCHEMA equilibrium decision-making (CLIENT INTERFACE SHELL).

The objective of (privacy-aware) network slicing is to maximize the performance of network slices in the system, while at the same time, fulfilling the privacy SSLAs requirements.

The objective of the network slicing can be expressed as:

$$\max_{\{x_{i,j}^{(t)}\}} \lim_{\tau \rightarrow \infty} \frac{1}{\tau} \sum_{t=0}^{\tau} \sum_{i \in I} \sum_{j \in J} U_{i,j}^{(t)}$$

As $\tau \rightarrow \infty$, the slicing orchestration problem is an infinite time horizon stochastic dynamic programming (optimization) problem.

5.2.4 Preliminary Results

Privacy-aware Orchestrator. We introduced (in the previous dry run) the Auction Mechanism. It actually consists of a fully scalable system that enables intra-domain containerized applications migration in a distributed multi-domain network, alongside privacy-aware slicing (already supported). We envisioned, in this demonstration setup, an overview of the multi-domain and distributed Auction Mechanism, where all local domains can orchestrate containers-services in parallel and exchange containers only during an auction. The following illustrations are real-case simulation scenarios and validation deriving results from the SCHEMA capabilities. Since scalability is not an issue for the Orchestrator, we plan to deploy in the next Release an even larger number of end nodes for the 5G/6G network showcase. Finally, we care about service latency as well as dissemination efforts to benchmark 6G Privacy KPIs, in future releases, among other Cybersecurity metrics like user/service/UE geo-location privacy preservation from the containerized applications migration, or even how *protective* is our SOAR-Security Orchestrator Automation and Response component, that provisions the whole stack of both SCHEMA and the rest functionality components and is able to become the main trusted authority of the (Privacy-aware) service function chaining operations.

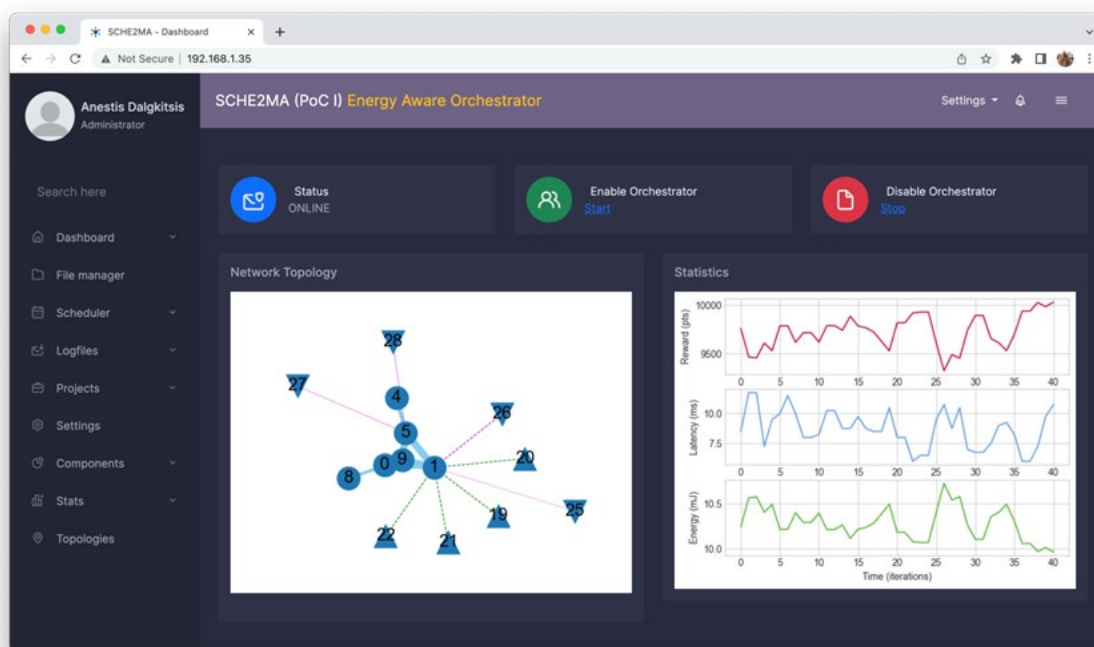


Figure 23 - SCHEMA demo dashboard from the experimentation plane.



Figure 24 - SCHEMA and its UI back-end running.

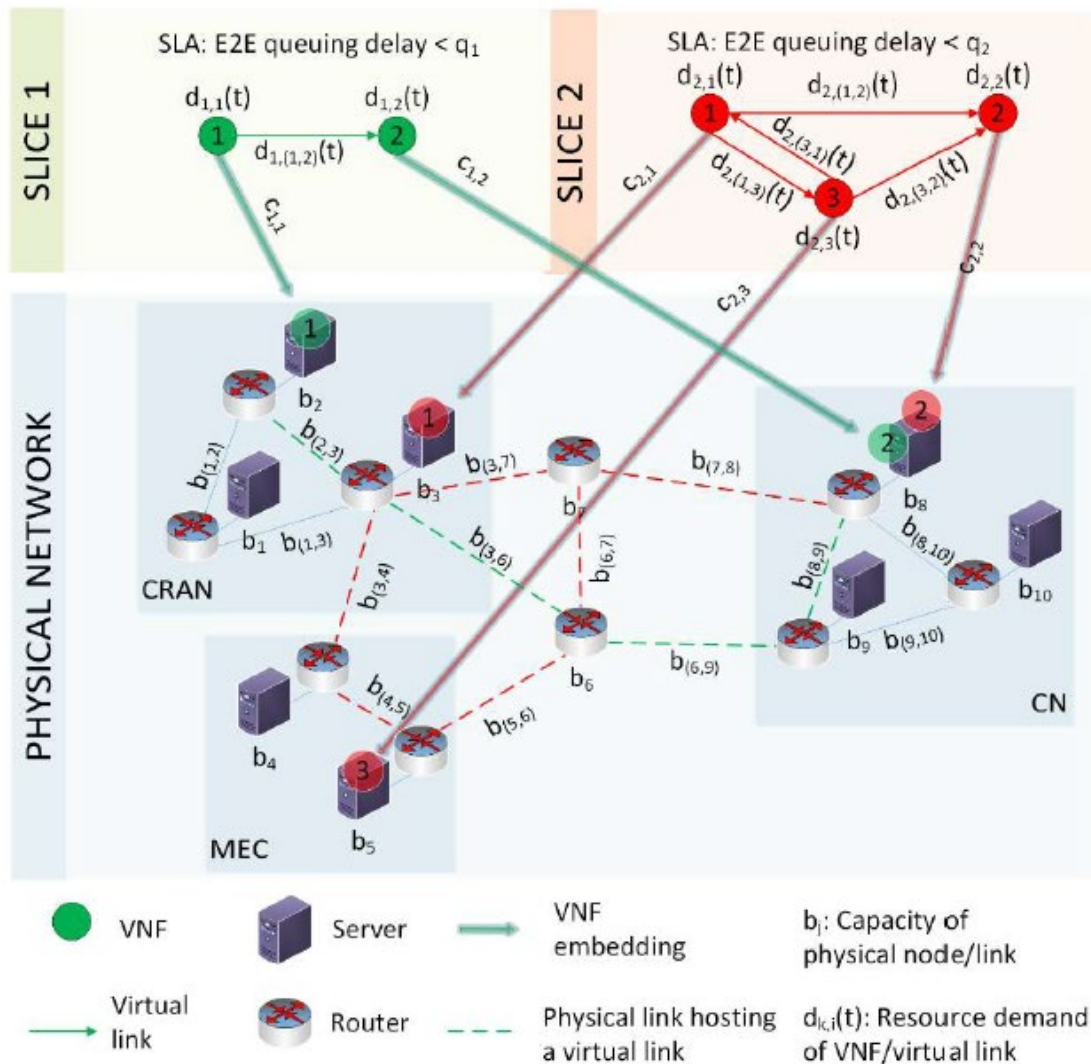


Figure 25 - A conceptual model of the SCHEMA system model capabilities to architecture and export VNF service graphs.

The containerized applications migration technique has been expanded in the following ways since it was first shown in this demo workflow for a single domain: The scheme can now support multiple containers per slice, including an arbitrary, probabilistic network functions service graph that permits loops. Additionally, it can support a range of realistic end-to-end performance metrics for the average flow served by such a container(s) graph. Lastly, there is a multi-agent solution that involves the presence of a DE (agent) with each slice, or even with each container of a slice, greatly enhancing the scheme's scalability as the number of slices (or containers-services host locations) increases while maintaining near-optimal performance.

Finally, this time, the probing approach for containerized services bottleneck localization is used in a more useful use case that depends on a Service Chaining Function that runs on Docker/Kubernetes.

5.3 FPGA attestation

In addition to software infrastructure and container attestation, PRIVATEER incorporates mechanisms for **hardware attestation** specifically designed for **Field-Programmable Gate Arrays (FPGAs)** deployed on edge nodes. PRIVATEER envisions that each node within the infrastructure provider's network is equipped with dedicated FPGA accelerators. These FPGAs serve to accelerate critical monitoring analytics tasks developed under **Task 3.5**. While these attestation mechanisms are not yet integrated into **Release A** of the PRIVATEER framework, they are planned for inclusion in **Release B**. During Release B, the attestation results from the FPGA attestation mechanism will be propagated and incorporated into the Level of Trust assessment mechanism. This will provide an extra factor for assessing the trustworthiness of each individual node within the network. Essentially, the attestation results will serve as an additional knob to fine-tune trust evaluation.

While FPGAs offer advantages like low latency and power efficiency, they are susceptible to various security attacks that can compromise the system's privacy (e.g. stealing/eavesdropping valuable information, loading malicious code, reverse engineering user's code etc.). To tackle this challenge, as described in D5.1, in PRIVATEER various methodologies are developed to provide the necessary security countermeasures for the hardware accelerators. Precisely, we primarily rely on custom remote attestation protocols, to ensure secure programming of such devices, as well as to protect the application and the infrastructure provided. In this workflow, apart from ensuring the secure programming of the hardware accelerators, a distribution of the security state for this service is shared with additional PRIVATEER components, i.e. the Blockchain. This enables the opportunity, in a future state of the project (after Release A), to incorporate the attestation results of the hardware accelerators into the LoT.

In the following section, the security countermeasures for hardware accelerators that will be showcased in this workflow are described.

FPGA Attestation protocol description

The core of the developed features a remote attestation protocol that is based on the one described in D5.1. Moreover, it involves the interaction among four entities: i) the User, ii) the Attestation Server iii) the Edge Accelerator and finally iv) Blockchain. Specifically, in the context of PRIVATEER:

(i) The *User* corresponds to the developer of the hardware accelerators for the anomaly detection AI models.

(ii) The attestation server represents an external server used for verification, that is responsible for interacting with all the parties involved for the integrity verification of the edge accelerators, including the User, the Edge server hosting the accelerator, as well as the PRIVATEER's blockchain.

(iii) The Edge server contains a x86-based CPU along with an FPGA card (i.e. Alveo family from AMD) connected via PCI-E, that serves as the hardware accelerator. Additionally, the system's host CPU is responsible for performing the security operations required.

(iv) The Blockchain refers to the infrastructure developed by a different Task of PRIVATEER.

The detailed steps of the developed remote attestation protocol are illustrated in Figure 26 and can be divided into 3 main steps:

1. An offline phase, which includes the preparation from the user, namely the encryption of the application from the developer, as well as uploading the reference values (i.e. the application's checksum) to a secure location. The developer user is also responsible for transferring the encrypted application to the edge server, with a generic secure transfer protocol. Furthermore, during the preparation stage, the operator of the Edge Server is also responsible for acquiring the reference values for the attestation service that will be deployed on the server.
2. Then, upon a request from the User/Developer, the remote attestation process instantiates and verifies two components: 1) the attestation service running on the Edge Server and 2) the accelerator kernel prior to being loaded in the FPGA. This is performed by generating in the Edge Server the attestation report, containing the respective checksum values, as well as random nonces, that is used for mitigating any possible replay attacks. Then, this report is transferred back to the verification server for checking all the individual values.
3. Upon a successful remote attestation (i.e. all the received values from the attestation report match with the reference data), the application's decryption key is delivered securely to the Edge server. Afterwards: (a) decryption of the application is executed and (b) the programming of the FPGA accelerator is performed. Furthermore, the outcomes of the attestation are forwarded to the Blockchain.

Providing a more detailed analysis of the proposed remote attestation protocol, the process initiates with the operator of the Edge server acquiring the reference values for the attestation service that will be deployed on the Edge server (**Step 1**), as well as uploading them to the Attestation Server (**Step 2**). Additionally, the preparation from



the user side includes the encryption of the application from the Developer (**Step 3**), as well as uploading the reference values (i.e. the application's checksum) also to the Attestation Server (**Step 4**). The user is also responsible for transferring the encrypted application to the edge server, prior to invoking for an attestation request. Following this preparation, the user can initiate a request for remote attestation (**Step 5**). After securing a connection with the Attestation Server and sending the request along with a randomly generated nonce (N1) to prevent replay attacks (**Step 6**), the Attestation Server receives the checksum of the attestation application and the nonce (**Step 7-8**). This is done to confirm that the correct verification service is operating on the Edge Accelerator (**Steps 9-10**). We note that, for verifying any received values, the Attestation Server checks with the reference values acquired in **Steps 2,4**. After a successful authentication, the Attestation Server then sends an attestation request to the Edge Accelerator, including a new randomly generated nonce (N2) (**Step 12-13**). The Edge Accelerator then calculates the static checksum of the received bitstream and retrieves the included bitstream certificate to generate the AttestReport. Additionally, the generated AttestReport is encrypted with a private key that is stored in the Edge Server (**Step 14**). Subsequently, the EncAttestReport is sent to the External Server for inspection (**Step 15**), in which, after decrypting the received values with the respective key, verification of each individual values is performed (**Step 16-17**). Then, the attestation results (successful/failed) are uploaded to the Attestation Server to the Blockchain (**Step 18**). Upon successful authentication of the received AttestReport, the verification server transfers the bitstream decryption key (BitstrDecKey) from the User to the Edge Accelerator using a secure key exchange algorithm, i.e. Elliptic Curve Diffie-Hellman (ECDH) (**Steps 19-20**). Once the Edge Server receives the Decryption Key, it decrypts the accelerator kernel and loads it into the Hardware Accelerator (**Step 21**).

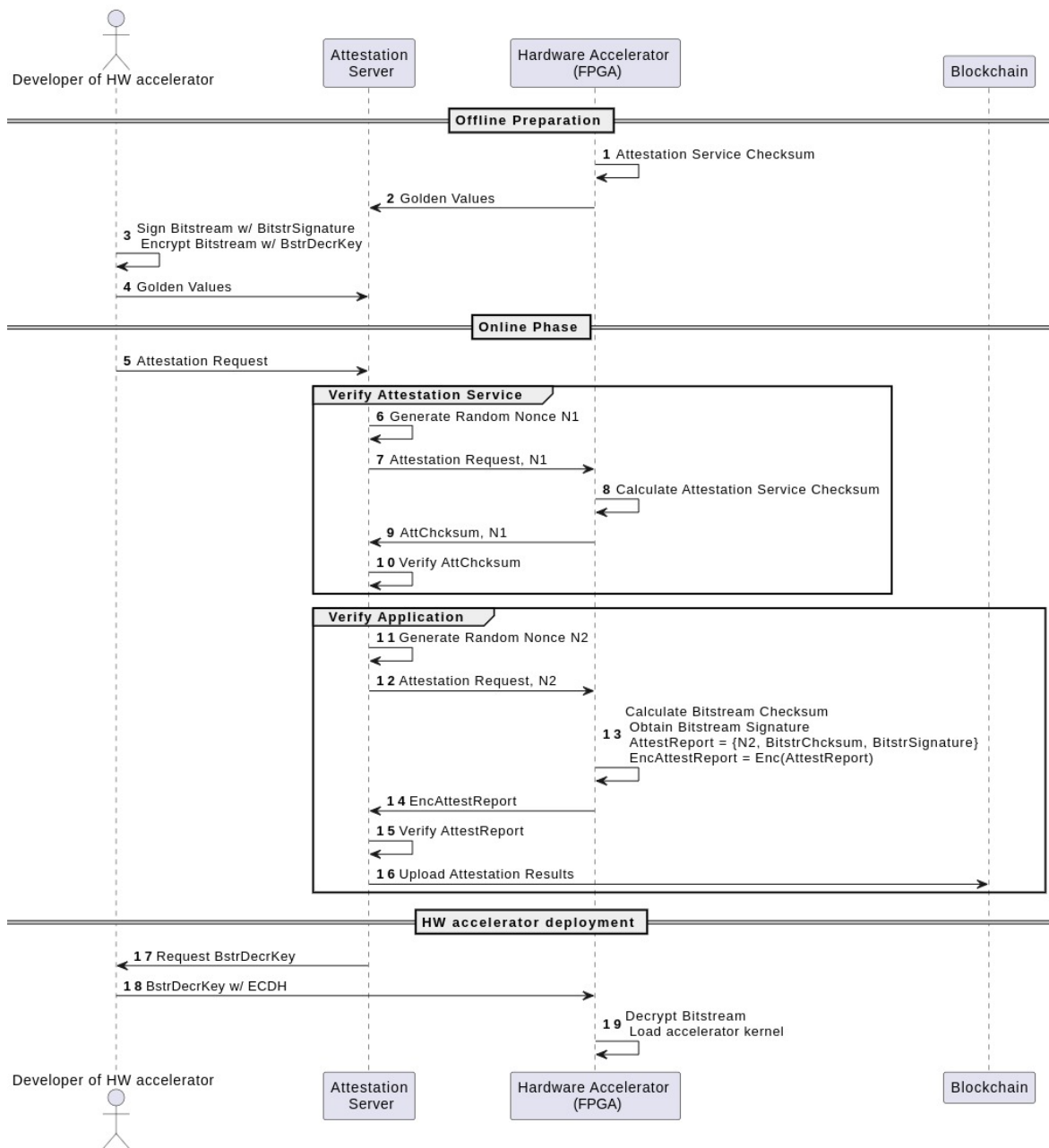


Figure 26 Remote attestation protocol for Workflow 5.3

5.3.1 APIs – Interfaces

The Attestation Server, as described in the previous sections, communicates with the PRIVATEER’s Blockchain, to upload the attestation results. The attestation server for the project’s Release A is hosted by ICCS, therefore a VPN connection is required to connect to the infrastructure provided in the NCSR facilities, where the Kafka broker is operating. We also note that, the attestation results that are forwarded to the Blockchain are structured in a predefined Json format.



5.3.2 Preliminary Results

In Figure 28 to Figure 30 some screenshots are presented, that showcase the functionality of the proposed remote attestation protocol, with the detailed transactions between the Edge Server and the Attestation Server.

The listed execution results, cover both successful and failed attempts to program the Edge Accelerator, to evaluate the developed methodologies under different circumstances. In Figure 28 and Figure 27 a successful attestation scenario is presented. In the first part (1b,2a), the verification of the infrastructure is performed, i.e. the attestation service running in the Edge Server. After a successful attestation, the protocol proceeds with the verification of the accelerator kernel (1c,2b). In the final sections (2c,2d) where the attestation succeeds, we observe that the bitstream is decrypted and loaded successfully to the accelerator.

Furthermore, from the last step of the Attestation Server (1d), we observe that the attestation evidence is uploaded to the Blockchain, through the Security Context Broker (SCB).



```

Edge Server w/ Hardware Accelerator

ipapal@coroni:~/Projects/Privateer/remote_attestation_example$ python3 client_socket_ssl.py
-----
Edge Accelerator connected to 147.102.37.120 [Port: 6666]
-----
..... Verifying attestation service .....
Received Nonce: c90444d9966b7184
-----
Sending Attestation report to the Verification Server...
c90444d9966b71843c5b64243fcee267023535589de4eefdb0b48e6b5ba70e9b253c759f28acf22a3
-----
Waiting for response...
✓ [Infrastructure] Successful Attestation
-----
..... Verifying accelerator kernel .....
Input file: app_files/hello_world_kernel/vadd_enc_signed.xclbin
Received Nonce: a6cc6bb8c236b128
-----
Bitstream Checksum: 2afe2f2a9bd509bba2e72e4a10d9cb4a49318dc06517a244cf66b598d74c49e6
Bitstream Signature: f8e2a7b1d6934c0f9dc5450e76a91b6e5e257db4c52e9f062d2464937d3a1c99
-----
Sending Attestation report to the Verification Server...
0332b8e6d6847c762a4f6433cdf837cf64c715d988ba87516362a741b3c6dc15e84b065bbf5c2aee6af5451a593f3d7acd5e1ee7e030578379bd4e0efa08b709f154
9bebf2ad7720c7c954dee131fb3e21c7613958e95cd50fb665e0b4e0501f687432ea43021cc0aebaf93b03af6e18196b240509f6ad01a5a2e7ee68f94abda42ede0a
ae1684d2b8be8cc9d88b
-----
Waiting for response...
✓ [Kernel] Successful Attestation
-----
Getting bitstream decryption key from the server using ECHD...
Key Derivation Completed
-----
Decrypting bitstream...
Building the .xclbin file...
Cleaning files...
-----
Loading the application to the accelerator...
Attestation Execution time: 1.65 sec
-----
Found Platform
Platform Name: Xilinx
INFO: Reading app_files/hello_world_kernel/output.xclbin
Loading: 'app_files/hello_world_kernel/output.xclbin'
Trying to program device[0]: xilinx_u50_gen3x4_xdna_base_2
XRT build version: 2.11.0
Build hash: 73a310b6e65651f4056a02d23f403883d5517a0f
Build date: 2024-04-08 16:52:26
Git branch: 2021.1
PID: 35139
UID: 1037
[Mon Jul 8 05:22:22 2024 GMT]
HOST: coroni
EXE: /home/ipapal/Projects/Privateer/remote_attestation_example/app_files/hello_world_kernel/hello_world
[XRT] ERROR: Xclbin does not match shell on card.
[XRT] ERROR: Shell VBNV is 'xilinx_u50_gen3x4_xdna_base_2'
[XRT] ERROR: Xclbin VBNV is 'xilinx_u200_xdna_201830_2'
[XRT] ERROR: Use 'xbmgmt flash' to update shell.
[XRT] ERROR: See dmesg log for details. err=-9
[XRT] ERROR: failed to load xclbin: Operation not supported
Failed to program device[0] with xclbin file!
Trying to program device[1]: xilinx_u200_xdna_201830_2
Device[1]: program successful!
TEST PASSED

```

Figure 27 - Successful attestation scenario with programming the hardware accelerator (Edge Server side).



```

Attestation Server
lighthouse :: Research Projects/Privateer/remote_attestation_example (master*) % python3 server socket ssl.py
[INFO:kafka.conn:<BrokerConnection node_id=bootstrap-0 host=10.160.3.213:9092 <connecting> [IPv4 ('10.160.3.213', 9092)]>: connecting to 10.160.3.213:9092 [IPv4 ('10.160.3.213', 9092) IPv4]
[INFO:kafka.conn:Probing node bootstrap-0 broker version
[INFO:kafka.conn:<BrokerConnection node_id=bootstrap-0 host=10.160.3.213:9092 <connecting> [IPv4 ('10.160.3.213', 9092)]>: Connection complete.
[INFO:kafka.conn:Broker version identified as 2.5.0
[INFO:kafka.conn:Set configuration api_version=(2, 5, 0) to skip auto check_version requests on startup
-----
Remote Attestation Server
Server listening on 147.102.37.120 [Port: 6666]

----- Infrastructure attestation -----
Nonce: c90444d9966b7184
-----
Received Attestation Report
Nonce : c90444d9966b7184
Checksum : 3c5b64243fcee267023535589de4eefd0b48e6b5ba70e9b253c759f28acf22a3
-----
Reference Values
Checksum : 3c5b64243fcee267023535589de4eefd0b48e6b5ba70e9b253c759f28acf22a3
-----
Attestation result:
✓ [Infrastructure] Successful Attestation
----- Accelerated kernel attestation -----
Nonce: a6cc6bb8c236b128
-----
Received Attestation Report
Nonce : a6cc6bb8c236b128
Checksum : 2afe2f2a9bd500bba2e72e4a10d9cb4a49310dc06517a244cf66b598d74c49e6
Certificate : f8e2a7b1d6934c0f9dc5450e76a91b6e5e257db4c52e9f062d2464937d3a1c99
-----
Reference Values
Checksum : 2afe2f2a9bd500bba2e72e4a10d9cb4a49310dc06517a244cf66b598d74c49e6
Certificate : f8e2a7b1d6934c0f9dc5450e76a91b6e5e257db4c52e9f062d2464937d3a1c99
-----
Attestation result:
✓ [Kernel] Successful Attestation

Sending the bitstream decryption key using ECDH...
Key Derivation Completed
[INFO:kafka.conn:<BrokerConnection node_id=1 host=10.160.3.213:9092 <connecting> [IPv4 ('10.160.3.213', 9092)]>: connecting to 10.160.3.213:9092 [IPv4 ('10.160.3.213', 9092) IPv4]
[INFO:kafka.conn:<BrokerConnection node_id=1 host=10.160.3.213:9092 <connecting> [IPv4 ('10.160.3.213', 9092)]>: Connection complete.
[INFO:kafka.conn:<BrokerConnection node_id=bootstrap-0 host=10.160.3.213:9092 <connected> [IPv4 ('10.160.3.213', 9092)]>: Closing connection.
[debug] topic : evidence.attestation
[debug] partition : 0
[debug] offset : 1007
[INFO:kafka.producer.kafka:Closing the Kafka producer with 9223372036.0 secs timeout.
[INFO:kafka.conn:<BrokerConnection node_id=1 host=10.160.3.213:9092 <connected> [IPv4 ('10.160.3.213', 9092)]>: Closing connection.

Remote Attestation Server
Server listening on 147.102.37.120 [Port: 6666]

```

Figure 28 - Successful attestation scenario with uploading attestation evidence to the Blockchain (Attestation Server side).

Apart from the successful attestation example, in Figure 29 and Figure 30, two different failed attestation scenarios are presented. In the first one (Figure 29), we notice that the cause of the failed attestation is the unverified attestation service in the Edge Server due to receiving the wrong checksum. Regarding the second unsuccessful attestation example (Figure 30), the accelerated kernel is the cause of failing, since the wrong checksum is received. In both cases, we observed that neither the hardware accelerator was programmed, nor the AI kernel was decrypted.



```
lpapal@coroni:~/Projects/Privateer/remote_attestation_example$ python3 client_socket_ssl.py
-----
Edge Accelerator connected to 147.102.37.120 [Port: 6666]
-----
----- Verifying attestation service -----
Received Nonce: cb496524cace6a51
-----
Sending Attestation report to the Verification Server...
cb496524cace6a519e267011c87e6a2ea56d456f6ec258611c527e3c3876c8a0e975b864401c7155
-----
Waiting for response...
X [Infrastructure] Failed Attestation
Exiting...

lighthouse :: Research_Projects/Privateer/remote_attestation_example <master* > % python3 server_socket_ssl.py
INFO:kafka.conn:<BrokerConnection node_id=bootstrap-0 host=10.160.3.213:9092 <connecting> [IPv4 ('10.160.3.213', 9092)]>: connecting t
o 10.160.3.213:9092 [('10.160.3.213', 9092) IPv4]
INFO:kafka.conn:Probing node bootstrap-0 broker version
INFO:kafka.conn:<BrokerConnection node_id=bootstrap-0 host=10.160.3.213:9092 <connecting> [IPv4 ('10.160.3.213', 9092)]>: Connection c
omplete.
INFO:kafka.conn:Broker version identified as 2.5.0
INFO:kafka.conn:Set configuration api_version=(2, 5, 0) to skip auto check_version requests on startup
-----
Remote Attestation Server
Server listening on 147.102.37.120 [Port: 6666]
-----
----- Infrastructure attestation -----
Nonce: cb496524cace6a51
-----
Received Attestation Report
Nonce : cb496524cace6a51
Checksum : 9e267011c87e6a2ea56d456f6ec258611c527e3c3876c8a0e975b864401c7155
-----
Reference Values
Checksum : b3f23fd7e20df63452b4b88a14d1ae90d97fce2673bf9f1ee837a9423c804718
-----
Attestation result:
X [Infrastructure] Attestation failed
Exiting...
```

Figure 29 - Failed attestation due to invalid attestation service (Top: Edge Accelerator, Bottom: Attestation Server)



```

ipapal@coroni: ~/Projects/Privateer/remote_attestation_example
ipapal@coroni:~/Projects/Privateer/remote_attestation_example$ python3 client_socket_ssl.py
Edge Accelerator connected to 147.102.37.120 [Port: 6666]
----- Verifying attestation service -----
Received Nonce: af5aab2cc9b4816c
-----
Sending Attestation report to the Verification Server...
af5aab2cc9b4816cb3f23fd7e20df63452b4b88a14d1ae90d97fce2673bf9f1ee837a9423c804718
-----
Waiting for response...
✓ [Infrastructure] Successful Attestation
----- Verifying accelerator kernel -----
Input file: app_files/hello_world_kernel/vadd_enc_signed.xclbin
Received Nonce: ea84f46cc83bc62
Bitstream Checksum: 2afe2f2a9bd500bba2e72e4a10d9cb4a49310dc06517a244cf66b598d74c49e6
Bitstream Signature: f8e2a7b1d6934c0f9dc5450e76a91b6e5e257db4c52e9f062d2464937d3a1c99
-----
Sending Attestation report to the Verification Server...
d0c5fcf5905ed0f8731130137f42bb72c64c715d980ba87516362a741b3c0dc15e84b65b5bf5c2aee6af5451a593f3d7acd5e1ee7e030578379bd4e8efa08eb709f154
9bebf2ad7720c7c954de6131fb3e21c7613958e95dc50fb665e0b4e0501f687432ea93021cc0aebaf95b03af6e18196b240509f6ad01a5a2e7ee68f94abda42edee0a
a61684d2b8e8cc9d08b
-----
Waiting for response...
✗ [Kernel] Failed Attestation
Exiting...
-----

```

```

python3 server_socket_ssl.py
Lighthouse :: Research_Projects/Privateer/remote_attestation_example (master*) % python3 server_socket_ssl.py
INFO:kafka.conn:<BrokerConnection node_id=bootstrap-0 host=10.160.3.213:9092 <connecting> [IPv4 ('10.160.3.213', 9092)]>: connecting to 10.160.3.213:9092 [('10.160.3.213', 9092) IPv4]
INFO:kafka.conn:Probing node bootstrap-0 broker version
INFO:kafka.conn:<BrokerConnection node_id=bootstrap-0 host=10.160.3.213:9092 <connecting> [IPv4 ('10.160.3.213', 9092)]>: Connection complete.
INFO:kafka.conn:Broker version identified as 2.5.0
INFO:kafka.conn:Set configuration api_version=(2, 5, 0) to skip auto check_version requests on startup
-----
Remote Attestation Server
Server listening on 147.102.37.120 [Port: 6666]
----- Infrastructure attestation -----
Nonce: af5aab2cc9b4816c
-----
Received Attestation Report
Nonce : af5aab2cc9b4816c
Checksum : b3f23fd7e20df63452b4b88a14d1ae90d97fce2673bf9f1ee837a9423c804718
-----
Reference Values
Checksum : b3f23fd7e20df63452b4b88a14d1ae90d97fce2673bf9f1ee837a9423c804718
-----
Attestation result:
✓ [Infrastructure] Successful Attestation
----- Accelerated kernel attestation -----
Nonce: ea84f46cc83bc62
-----
Received Attestation Report
Nonce : ea84f46cc83bc62
Checksum : 2afe2f2a9bd500bba2e72e4a10d9cb4a49310dc06517a244cf66b598d74c49e6
Certificate : f8e2a7b1d6934c0f9dc5450e76a91b6e5e257db4c52e9f062d2464937d3a1c99
-----
Reference Values
Checksum : 2afe2f2a9bd500bba2e72e4a10d9cb4a49310dc06517a244cf66b598d74c49e7
Certificate : f8e2a7b1d6934c0f9dc5450e76a91b6e5e257db4c52e9f062d2464937d3a1c99
-----
Attestation result:
✗ [Kernel] Attestation failed
INFO:kafka.conn:<BrokerConnection node_id=1 host=10.160.3.213:9092 <connecting> [IPv4 ('10.160.3.213', 9092)]>: connecting to 10.160.3.213:9092 [('10.160.3.213', 9092) IPv4]
INFO:kafka.conn:<BrokerConnection node_id=1 host=10.160.3.213:9092 <connecting> [IPv4 ('10.160.3.213', 9092)]>: Connection complete.
INFO:kafka.conn:<BrokerConnection node_id=bootstrap-0 host=10.160.3.213:9092 <connected> [IPv4 ('10.160.3.213', 9092)]>: Closing connection.
[debug] topic : evidence.attestation
[debug] partition : 0
[debug] offset : 1009
INFO:kafka.producer.kafka:Closing the Kafka producer with 9223372036.0 secs timeout.
INFO:kafka.conn:<BrokerConnection node_id=1 host=10.160.3.213:9092 <connected> [IPv4 ('10.160.3.213', 9092)]>: Closing connection.
-----
Remote Attestation Server
Server listening on 147.102.37.120 [Port: 6666]
-----

```

Figure 30 Failed attestation due to invalid/unauthorized accelerator kernel.

Regarding the KPIs for this workflow, we focus on the additional time required (namely time overhead) to program the hardware accelerators (i.e. FPGAs), when applying the developed methodologies for attesting these devices. This is essentially the time required for the Edge server to complete the attestation process. We aim for an attestation time of under 10 sec. We note that, from the reported screenshots, we observe that we obtain a successful attestation in under 2 sec with the current setup. However, we leave margins in the selected KPI, to be able to meet the target in the



future Release B, where the use of Intel SGX technology will be examined, running the remote attestation in lower end devices (with less computational power), as well as deploying security functions in the hardware accelerators. This feature adds supplementary overhead due to the time required for the programming of the device.



6 Workflow 2: DDoS detection by Federated NWDAF & CTI Sharing and SLA Management

6.1 Attack models

DDoS Attack

A Distributed-Denial-of-Service (DDoS) attack is a malicious attempt to disrupt the normal traffic of a targeted server, service, or network by overwhelming the target or its surrounding infrastructure with a flood of internet traffic. In the context of the 6G architecture, DDoS attacks have been identified as potential security risks, particularly for technologies such as Network-Function Virtualization (NFV), Software-Defined Wide-Area Networks (SD-WAN), Virtualized Radio-Access Networks (vRAN), Open RAN, and programmable HW Accelerators. This was documented in the deliverable '6G Threat Landscape and Gap Analysis' (D2.1), which highlighted the main technological innovations required for the envisioned 6G architecture and the potential cybersecurity threats they could face.

To accommodate for these risks, PRIVATEER partners have been working on releasing open-source datasets from the 5G+ infrastructure located in the premises of NCSR D capturing DDoS attacks initiated by malicious connected user equipment (UEs). This infrastructure will also serve as the testbed for deploying use-case scenarios for Release B of the project. The process of generating and utilizing this data has been thoroughly documented in D3.1 "Decentralised Robust Security Analytics Enablers Rel. A" [2].

Federated-Learning Attack Models

The attack surface of federated learning (FL) includes all the weaknesses that attackers can target. Typically, attackers break into the system by compromising the security of the central servers, local devices, or participants in the FL process. Once they gain access, attackers can change training parameters, model updates, and the results of the learning process. The taxonomy of FL attacks can be separated into two main categories, namely poisoning attacks and inference attacks. Poisoning attacks primarily threaten the robustness and integrity of the FL process, by introducing corrupted data or malicious updates that skew the learning process. On the other hand, inference attacks focus on compromising the privacy of FL participants by extracting sensitive information from the model's outputs or shared data during the training phase. These attacks have been thoroughly documented in D3.1 "Decentralised Robust Security Analytics Enablers Rel. A" [2].



Defining a comprehensive threat model in an FL scheme requires a structured approach that considers the different entities involved, their knowledge, capabilities and access to auxiliary data. The following steps help in identifying and characterizing these factors to ensure that robust security and privacy measures are in place.

1. Identify Key Entities

- Server: Coordinates the training process and aggregates updates.
- Participants (Clients): Perform local training and send updates to the server.
- Eavesdroppers: Unauthorized entities that intercept communications between the server and participants.

2. Classify Types of Participants

- Honest-but-Curious Participants: Follow the FL protocol but attempt to infer as much private information as possible from the data they have access to.
- Malicious Participants: Provide no guarantee of following the FL protocol and may engage in activities such as data poisoning, model poisoning, or attempting to extract private data from other participants.

3. Assess Knowledge

- Model Structure: Determine which entities have access to the model architecture.
- Weights and Gradients: Identify which entities can access the weights and gradients during training.

4. Evaluate Capabilities

- Training and Design: Can the entity train or redesign the model?
- Modifying Updates: Can the entity modify the updates sent to the server or to participants?

5. Consider Access to Auxiliary Data

- Server: Typically, does not have access to auxiliary data that can be used to infer additional information.
- Participants: Often have auxiliary data that can aid in performing inference attacks.
- Eavesdroppers: Generally, lack auxiliary data beyond intercepted communications.

In threat modelling for FL the severity of threats can vary significantly based on the number and behaviour of participants, their knowledge, and their capabilities. More malicious participants increase the risk, particularly if they possess extensive knowledge of the model structure, weights, and gradients, as well as access to

auxiliary data. Participants with advanced capabilities, such as the ability to modify updates or perform sophisticated inference attacks, pose higher threats.

In the context of WP3, we are going to implement privacy-preserving methods like Differential Privacy (DP) and Multi-Party Computation (MPC) to cater to the varying severity levels of threats in FL. These methods will help mitigate risks posed by both honest-but-curious and malicious participants, ensuring robust security and privacy. By incorporating these advanced techniques, we aim to protect against data breaches, model manipulation, and unauthorized inference attacks, thereby maintaining the integrity and confidentiality of the learning process.

6.2 Workflow 2 implementation

6.2.1 Short Description

This workflow showcases the PRIVATEER functionalities of anomaly detection through the security analytics deployed on the edge nodes, as well as the privacy-preserving CTI-sharing capabilities. Prior to workflow initiation, the security-analytics module must be installed on the PRIVATEER edge nodes. This module uses the Network-Data Analytics Function (NWDAF) to continuously monitor network traffic. The process begins when an attacker launches a DDoS attack targeted at these edge nodes. Obviously, one or more edge nodes can be affected by the attack. The machine-learning (ML) anomaly-detection model within the security-analytics module then detects any unusual behaviour and classifies the type of attack. After the detection, an alert is created and transmitted to the CTI-sharing proxy API, which creates a new event in the MISP instance of the entity that received the information. The users that have access to the CTI-sharing proxy API can then set up policies that define what can be shared and what the entity intends to search for in other entities. These policies contain the indicators of compromise (IoC) of interest as values. The proxy API allows the entity's users to set up a sharing group, to which other entities instances can be added to. This requires a peering process to be done first, where the public key of each entity is exchanged. Once this sharing group has multiple users, it is configured to connect to a reverse index database. This database uses trapdoors (a secure hash) of the IoCs as the index for each row of information, of which the value is also encrypted. The encrypted value contains a combination of the trapdoor signed and the Universally Unique Identifier (UUID) of the entity that uploaded the value. If an entity requests an update of the shared index, the proxy API checks the IoC's in the published events stored in the MISP and compares them to the policies of the entity. When there is a match, the shared index is updated with that information. Once an entity has already accessed the shared index and found that another entity has data on an IoC they're interested in, the entity gets in contact with that second entity and performs a synchronization of the MISP through the proxy API. First, it checks the policies of the requesting entity, ensuring there is an inward policy for the IoC and then sends a request for that event to the second entity. The entity that is storing the requested data checks if it has an outward policy for the requested IoC, ignoring the request if it does since outward policies act as a "blacklist" of IoC's that can't be shared. As long as there is no outward policy, the second entity accepts the first entity's request and returns the events containing the IoC. The first entity receives this information and then adds it to the MISP.

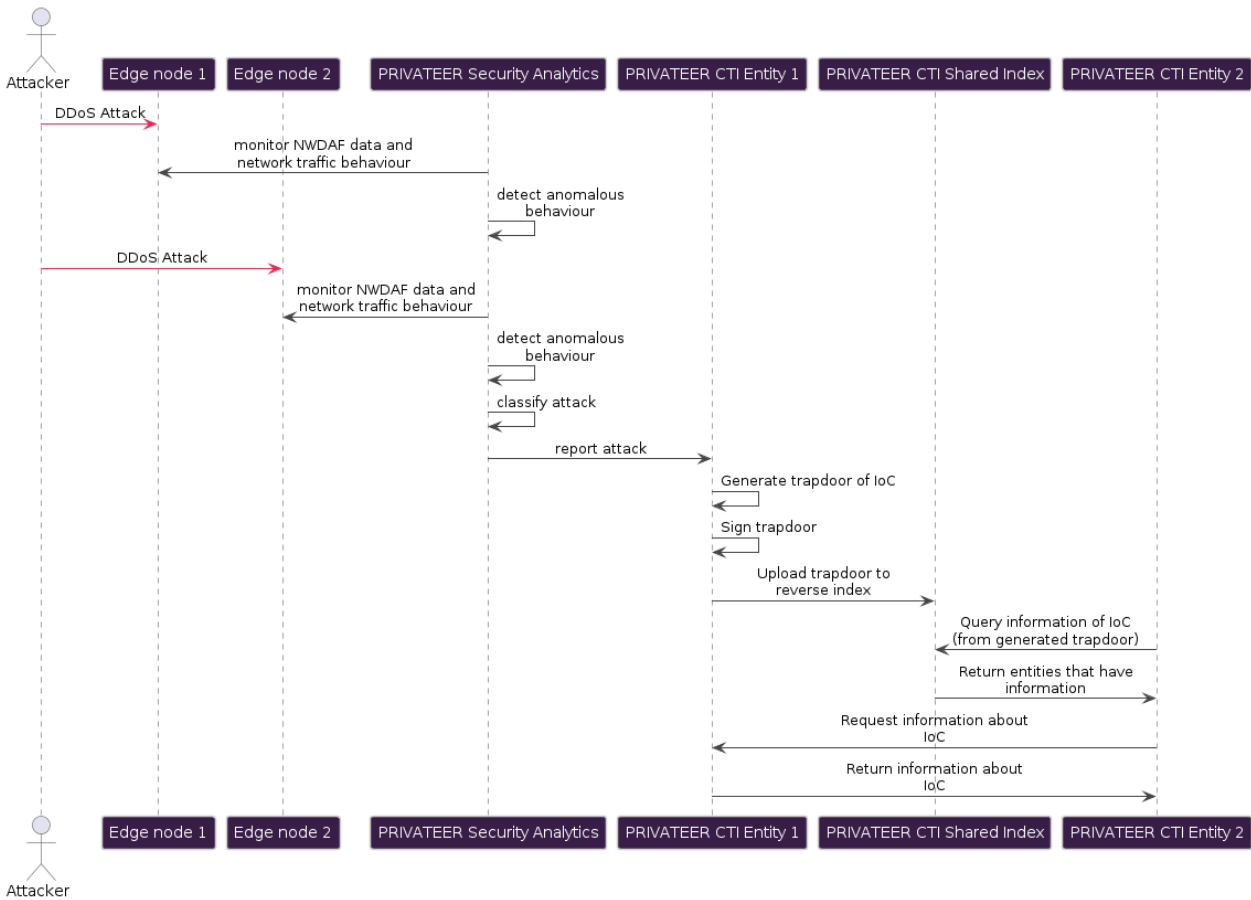


Figure 31 - Sequence diagram of DDoS-attack detection on two of the edge nodes and consequent CTI sharing.

6.2.2 Detailed Description

The workflow for "DDoS detection by Federated NWDAF and CTI Sharing" involves two primary stages: training and inference, each supported by the architecture depicted in Figure 32.

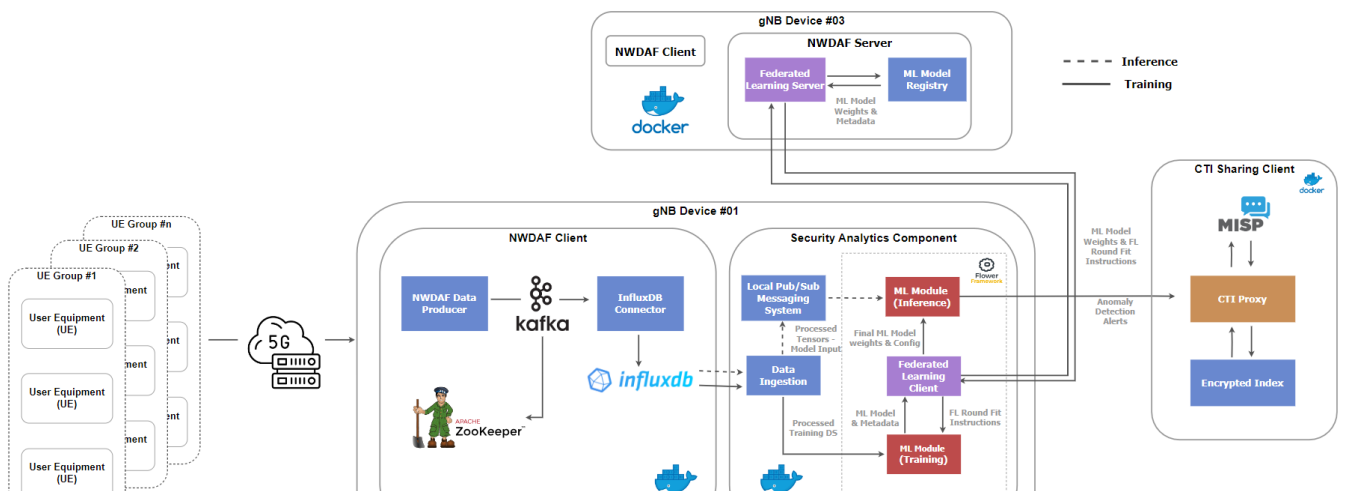


Figure 32 - Workflow 2 Architecture

UE devices are the primary sources of real-time data regarding their operational status, performance metrics, and connectivity quality. Each UE's interactions with the network, such as signal strength, data usage, and mobility patterns, are captured and streamed to Kafka through data producers deployed across gNB (gNodeB) devices. This continuous data from UEs provides the dataset for the NWDAF to analyze and optimize network performance.

The NWDAF API can interact with InfluxDB through the InfluxDB API to retrieve and analyse time-series data, enabling the NWDAF API to access detailed network metrics and performance indicators stored in InfluxDB. NWDAF consists of several key components that carry out the main functionality. ZooKeeper acts as the configuration and synchronization service for Kafka, which relies on ZooKeeper to maintain its state and manage cluster coordination, ensuring that the Kafka brokers are functioning properly and are synchronized. The Producer component extracts and transforms data received from gNB and sends it to Kafka. Kafka serves as the messaging backbone, receiving data from the Producer, which it then makes available to other components in the system. InfluxDB Connector extracts data from Kafka and writes it into InfluxDB, which stores the time-series data provided by the InfluxDB Connector. This database is optimized for handling large volumes of time-stamped data, making it suitable for analytics and monitoring tasks.

The deployment of a trained anomaly-detection model is essential for the effective detection of DDoS attacks within our workflow. In the context of the PRIVATEER framework, the training of such a model is implemented in a decentralized manner using an FL scheme. NWDAF data is consumed via a data-ingestion module, which is responsible for executing the necessary transformations to the original data to be further consumed by the ML model. During training, the module constructs the training and validation datasets necessary for the training process, while during inference it provides the data in the format needed by the inference module.

The FL clients are responsible for training local models on the data they process. This training is performed under the constraints and configurations dictated by the central FL Server, which manages the learning process across all clients. Once the local models are trained, the model updates—specifically, the parameters or weights of the models—are sent to the FL Server. After the central model has been updated, it is then sent back to the ML-Model Registry. This registry maintains the latest version of the model, ensuring that all clients can access the most current model for their ongoing operations. The updated model is subsequently redeployed across all participating NWDAF Clients. The FL Client and Server modules are implemented using the Flower Framework, which utilizes a server-client architecture that leverages gRPC protocol for communication between clients and the server.

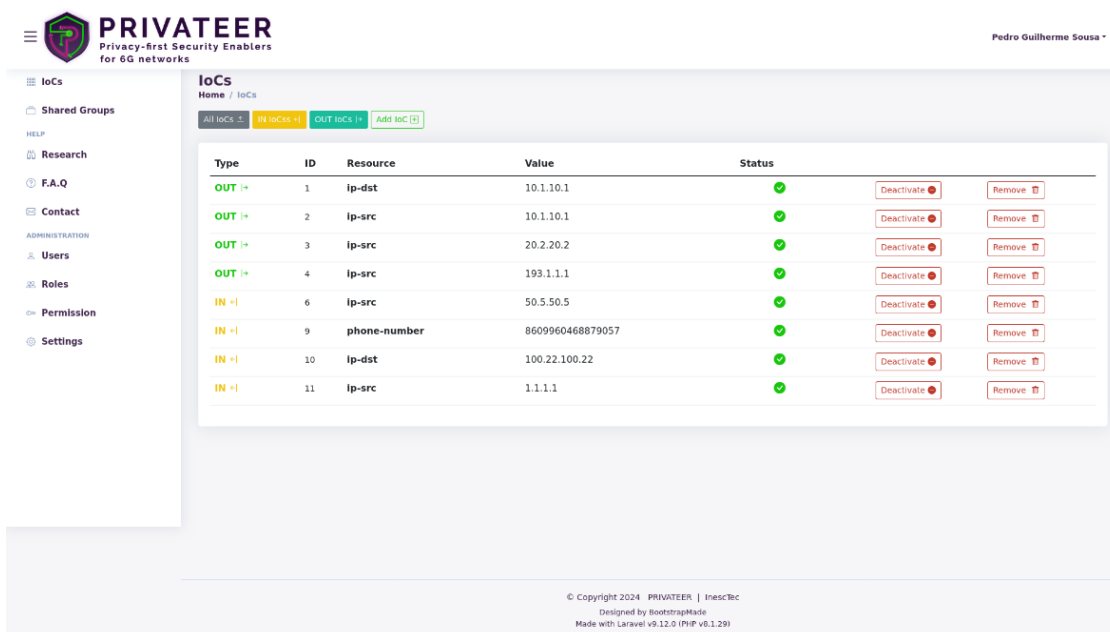
The workflow architecture includes also the CTI-Sharing Component, essential for effective and secure sharing of threat intelligence across entities. This component

requires three key elements for deployment by each participating entity: a MISP instance to store and organize IoCs, a CTI-sharing proxy API for managing the exchange of information between entities, and a database to host the shared index, storing UUIDs of entities that hold specific IoC information.

During inference, NWDAF data, processed by the data-ingestion module, are managed by a pub/sub-messaging system implemented using Redis. The ML Module (inference), equipped with the latest ML model distributed after the FL phase, consumes the pre-processed data by the messaging system and executes model inference. Upon detecting an anomaly, the workflow engages the CTI-sharing proxy API for alert management and dissemination. These alerts are routed through the CTI proxy to Reverse Index DB and MISP instances, ensuring a coordinated response to detected threats.

6.2.3 APIs/Interfaces

Figure 33 shows the home page of the interface of the CTI Sharing proxy API with the current policies registered in the database. The type, resource, value and status (whether the policy is active or not) are shown. Buttons that allow for the deactivation and removal of the policy are also available. The interface also provides a filter function, to choose between inward and outward policies.



The screenshot shows the PRIVATEER web interface. The header includes the PRIVATEER logo and the text "Privacy-first Security Enablers for 6G networks". The user name "Pedro Guilherme Sousa" is visible in the top right. The main content area is titled "IoCs" and contains a table of registered policies. The table has columns for Type, ID, Resource, Value, and Status. Each row includes "Deactivate" and "Remove" buttons. A sidebar on the left contains navigation links for "IoCs", "Shared Groups", "HELP", "Research", "F.A.Q", "Contact", "ADMINISTRATION", "Users", "Roles", "Permission", and "Settings".

Type	ID	Resource	Value	Status	Deactivate	Remove
OUT	1	ip-dst	10.1.10.1	Active	Deactivate	Remove
OUT	2	ip-src	10.1.10.1	Active	Deactivate	Remove
OUT	3	ip-src	20.2.20.2	Active	Deactivate	Remove
OUT	4	ip-src	193.1.1.1	Active	Deactivate	Remove
IN	6	ip-src	50.5.50.5	Active	Deactivate	Remove
IN	9	phone-number	8609960468879057	Active	Deactivate	Remove
IN	10	ip-dst	100.22.100.22	Active	Deactivate	Remove
IN	11	ip-src	1.1.1.1	Active	Deactivate	Remove

© Copyright 2024 PRIVATEER | InesTec
Designed by BootstrapMade
Made with Laravel v9.12.0 (PHP v8.1.29)

Figure 33 - User interface of CTI sharing module

All this information is retrieved with a GET request to the “/general/policies” route. The activation/deactivation of policies is achieved with a PUT request to “/policies/:id/activation” with the body containing a boolean variable “activated”.

Figure 34 shows that the interface also allows for users to create new policies. The popup window requests whether the policy will apply to inward or outward

communications, the type of IoC and its' value. This policy is added to the database of the proxy API through a POST request to “/general/policies”. The body of this request contains an object with a variable containing the type of IoC and its' value.

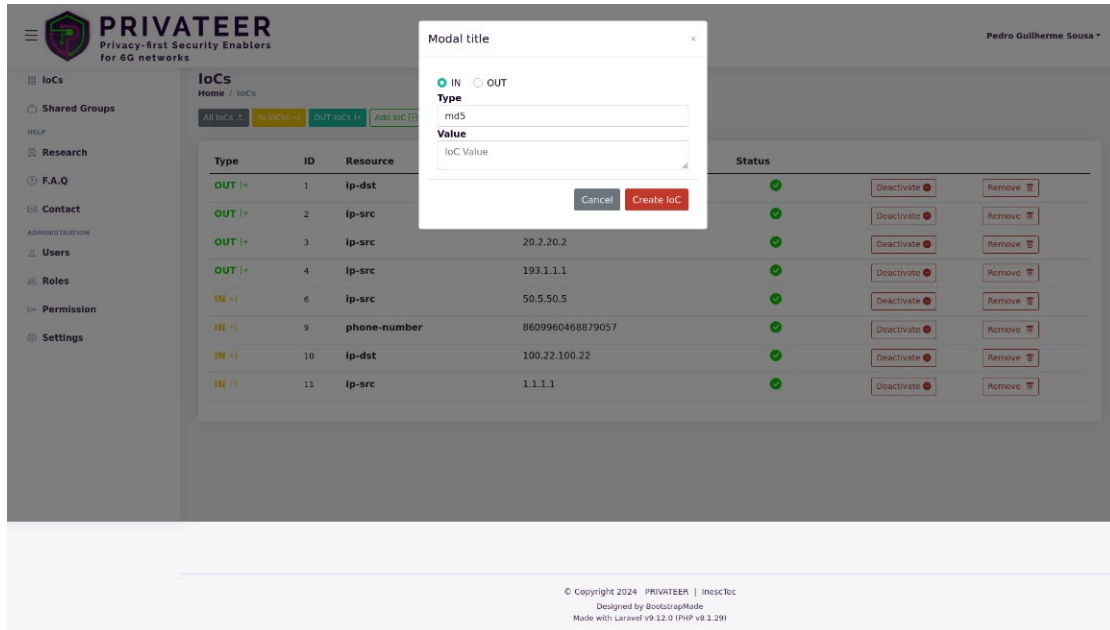


Figure 34 - Policy creation

The shared groups page shows the list of the sharing groups the current entity belongs to (see Figure 35). This is retrieved with a GET request to the “/shared-groups” route. More information about the sharing group can be viewed by clicking the “View” button. A request to update the shared index linked to a specific sharing group can be performed as well through the “index immediate update” button. This sends a GET request to the “/shared-groups/:id/index/immediate-update” route.

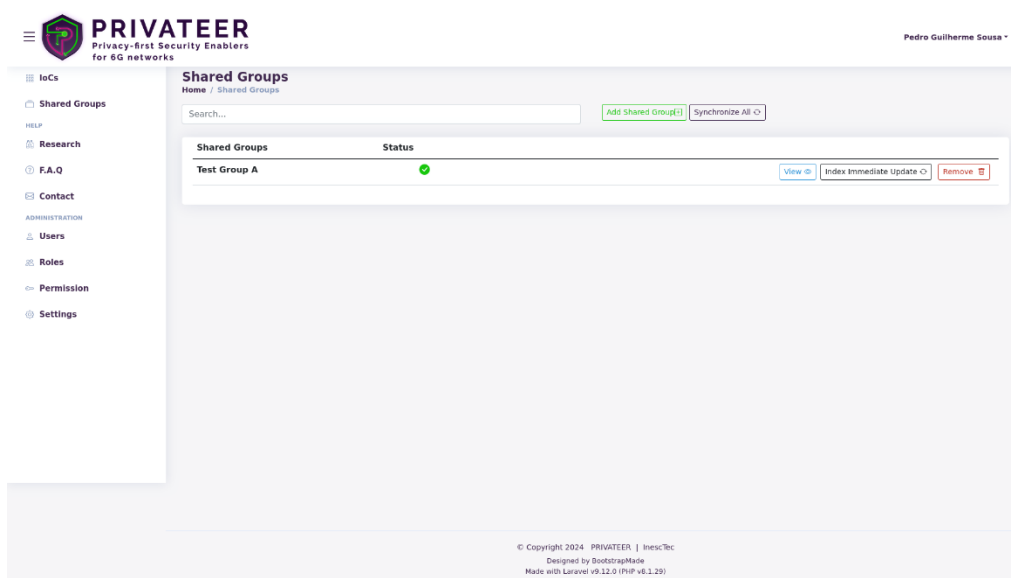


Figure 35 - Shared groups list

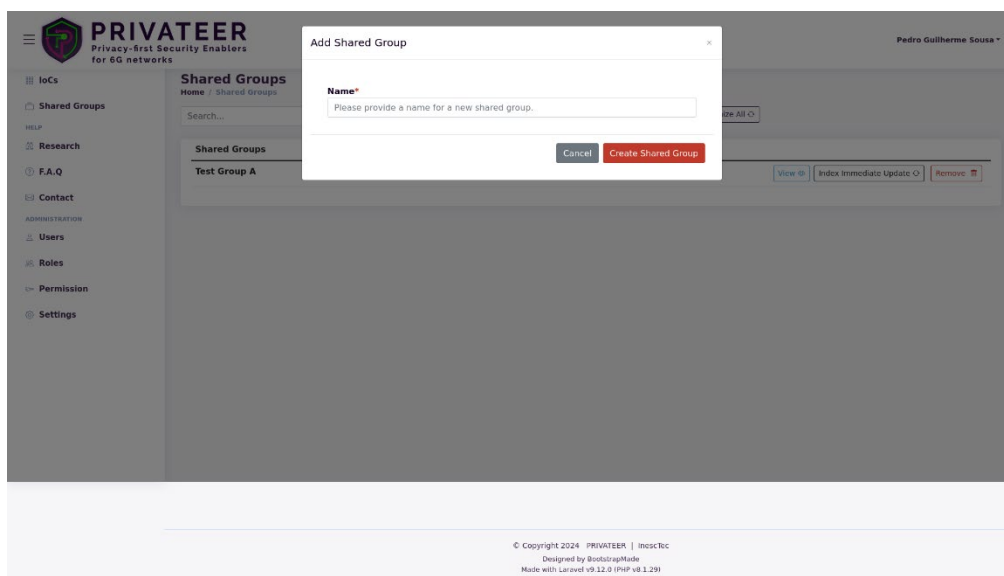


Figure 36 - Shared group creation

A new sharing group can be created here as well, as demonstrated in Figure 36. The only field necessary is a name. Once the group is created, entities can be added in the “View” page which displays more information. Shared groups are created by sending a POST request to the “/shared-groups” route.

The page with detailed information of the sharing group, shown in Figure 37, will present the other entities belonging to the sharing group. This information is retrieved with a GET request to the “/shared-groups/:id/entities” route. In this page a user can also request an update to the index associated to the sharing group but they can also check the connection with the other entities (which can be done by sending a GET request to the “/entities/:uuid/peer-sender” route and receiving a response without errors) and also request a synchronization of their data with the entity. This synchronization affects the MISP instance running on the current entity and will update or create any events that match the policies shown in Figure 33. This is done so by sending a GET request to the “/entities/:uuid/immediate-sync” route. An entity can be removed from a sharing group, but this requires that every participant does the same to ensure the sharing group works correctly.

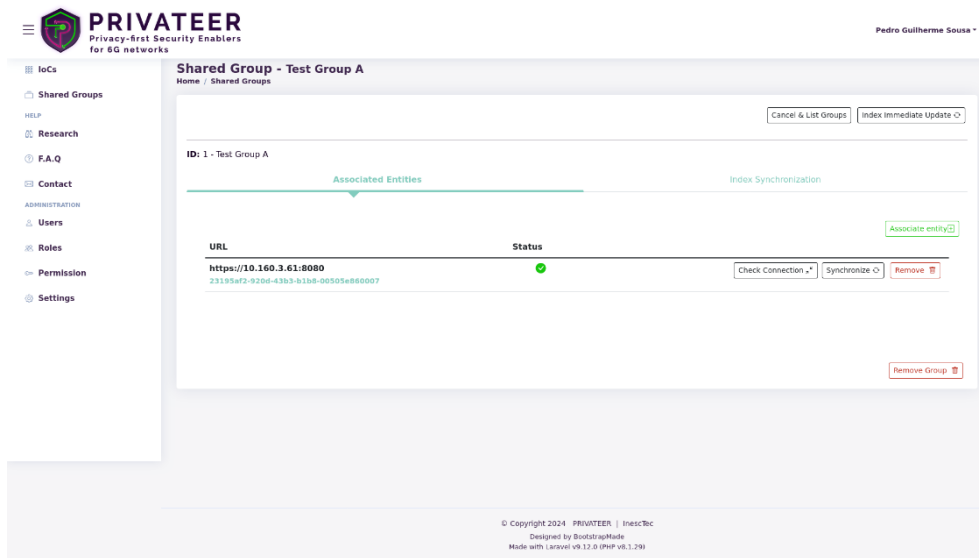


Figure 37 - Shared group details

Figure 38 shows how users can add a new entity to a shared group. The URL (in this case the IP address of the entity) of the entity is what is requested. Once the user inputs that information, the UUID of the entity that matches that URL is retrieved and the entity is linked, locally, to the sharing group by sending a POST request to the “/shared-groups/:id/entities” route with a body containing a variable “uuid” and the retrieved value. This requires that the entity is added to the system beforehand. It also requires that a peering process is completed beforehand as well. This is done through terminal instead and not available through the interface.

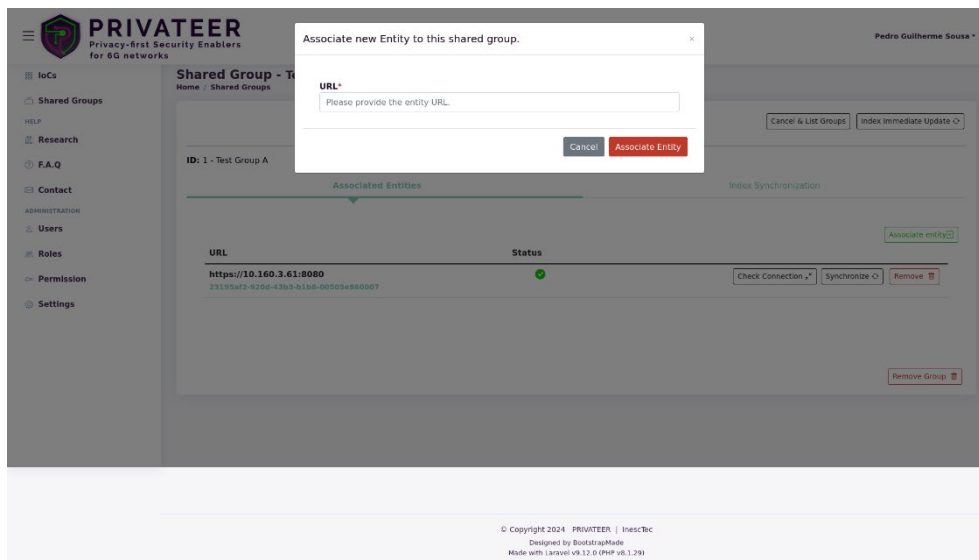


Figure 38 - Adding entity to shared group



6.2.4 Preliminary Results

While the implementation of the FL client and server is still in development, we leveraged Flower's simulation capabilities to emulate FL. The results, along with a comparison to the central model, are illustrated in **Σφάλμα! Το αρχείο προέλευσης της αναφοράς δεν βρέθηκε..** To generate federated datasets for each NWDAF client, IMEISVs were assigned to the NWDAF clients randomly.

Table 2 - Privateer Security Analytics Deep Learning Anomaly Detection model performance in centralized vs federated training

Metric	Centralized Learning	Federated Learning
Accuracy	0.9982	0.9939
Precision	1	0.9640
Recall	0.9822	0.9763
F1 Score	0.9910	0.9701
True Positives	996	990
True Negatives	8975	8938
False Positives	0	37
False Negatives	18	24

As observed, the model performance does not change considerably, as anticipated. This is due to the small number of clients (2), which causes the federated-training process to closely mirror the centralized approach.

6.2.5 Workflow 2 Demonstrators

6.2.5.1 Anomaly detection & IoC posting

In this section, we provide a comprehensive demonstration of the current developments in security analytics.

As depicted in Fig 31, the workflow 2 architecture includes two primary ML modules: one dedicated to inference and the other to training. Presently, the models are trained using a centralized approach. The FL client has not yet been implemented. For an in-depth explanation of the model-training process and the resulting outcomes, please refer to “Del 3.1 - Decentralised Robust Security Analytics Enablers Rel. A”.

Regarding the Local Pub/Sub messaging system, we have implemented Redis to facilitate communication. For data ingestion, we have developed a Python module designed to continuously interact with the NWDAF API. This module executes the following preprocessing steps:

- Extracts data from the NWDAF API response in a tabular format

- Applies smoothing techniques using a rolling average
- Generates the tensor input required by the model

Once processed, the data is appended to a Redis queue. The inference application then initiates a model instance, retrieves the next items from the Redis queue, and performs inference on these items. Upon detecting an anomaly, the system generates an alert as depicted in Figure 39.

```
45 def post_ioc(imeisv):
46     headers = {
47         "Accept": "application/json",
48         "Content-Type": "application/json"
49     }
50
51     event_data = {
52         "threat_level_id": "1",
53         "info": "Test Event 4",
54         "published": False,
55         "analysis": "0",
56         "distribution": "0",
57         "Attribute": [
58             {
59                 "type": "phone-number",
60                 "category": "Other",
61                 "value": imeisv
62             }
63         ]
64     }
65
66     response = requests.post(ALERT_API_URL, headers=headers, data=json.dumps(event_data), verify=False)
```

Figure 39 - Code snippet demonstrating the IoC generation and posting

The prerequisites for running the inference application include the model configuration, the python classes for the developed deep-learning model, and the model weights. For the data-ingestion application, a fitted scaler based on the training data is necessary. In Fig 39, the structure of the deployed repository is illustrated. The **data_collector** application handles the data ingestion module, while the **inference** application manages the machine learning inference module. We utilized Docker and Docker Compose to deploy the entire demonstrator system. API endpoints are configured as environment variables within the Docker images.

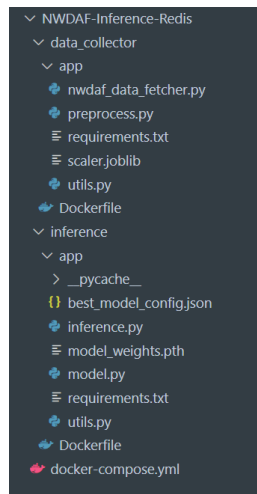


Figure 40 - Inference Demo repo structure

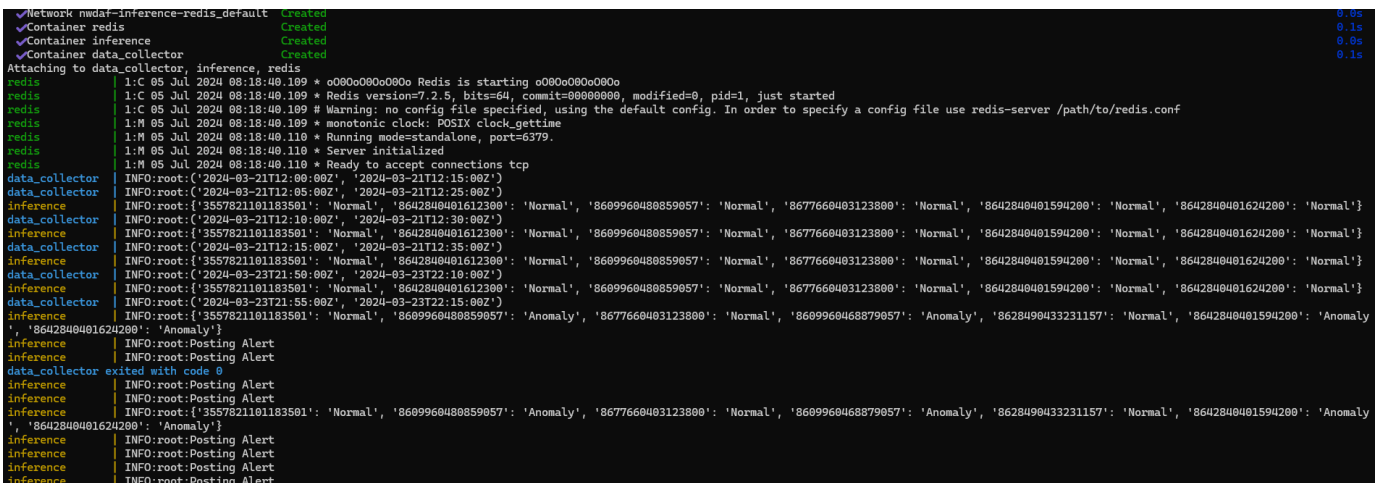


Figure 41 - Terminal view of the Workflow Demonstrator

Figure 41 provides a terminal view of the workflow. The **data_collector** module outputs the time ranges during which data is acquired via the NWDAF API. Concurrently, the **inference** module outputs the inference results for each IMEISV within each time range and indicates whether an alert has been raised if an anomaly is detected.

After an alert is generated and an IoC is posted, the IoCs become visible through the MISP instance. An example of this can be seen in Figure 42.

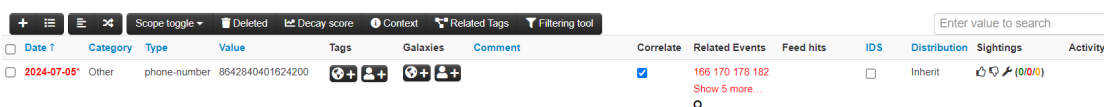


Figure 42 - IoC for imeisv 8642840401624200

6.2.5.2 XAI framework

The explainable Artificial Intelligence (XAI) is being developed in a standalone module which aims to provide insight into the decision-making process of the DDoS-detection algorithm used in workflow 2. Leveraging the DDoS-detection algorithm, this implementation uses optimized heuristics to produce a loss function able to provide details on how the decision-making process operates. This is done through the development of an XAI dashboard which uses the same dataset and model as described in workflow 2, Figure 43. Despite this, in its current form, this work occurs in parallel with the workflow 2 system and, although not yet fully integrated, it is able to provide an analysis of how the decision process is done by the security-analytics framework.

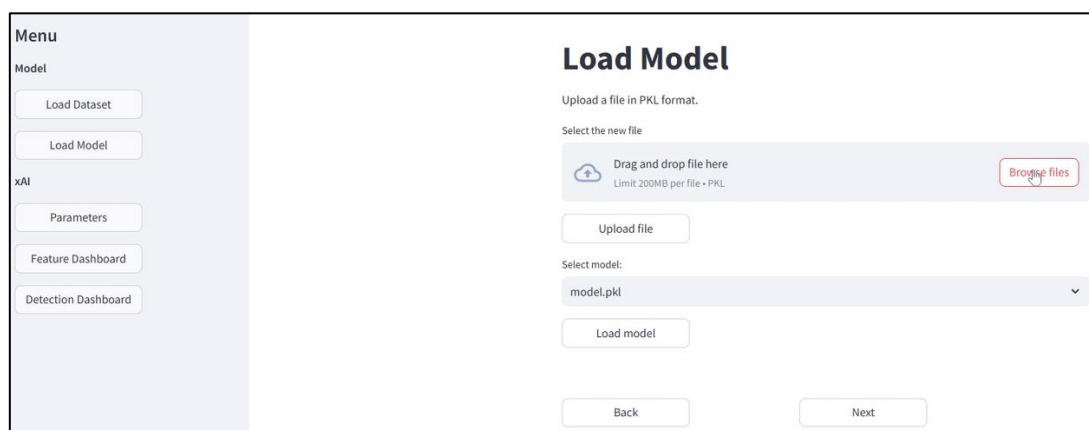


Figure 43 - XAI Framework for workflow 2

The current XAI techniques implemented are based on the SHAP framework, namely the SHAP Kernel Explainer which allows local instance explanation based on abstract deep-learning models. This approach still uses the detection algorithm and training timeseries datasets to initialize and operate its internal structure.

Timeseries require specialized XAI approaches to capture the temporal data dependencies between instances. Therefore, to capture time dependence between instances, the initial dataset is transformed so that each XAI instance contains information regarding the time windows in which detection is processed. This step was then combined with different heuristic strategies to provide multiple insights into the decision-making process.

In this release, the XAI module produces explanation based on 2 different heuristic strategies, as showed in Figure 44. Taking into consideration the nature of the detection algorithm, two different loss functions were strategized in order to capture feature loss and overall loss of the detection algorithm. This allows to probe which features are behaving anomalously with the nature of the detection algorithm and overall loss in DDoS-anomaly detection.

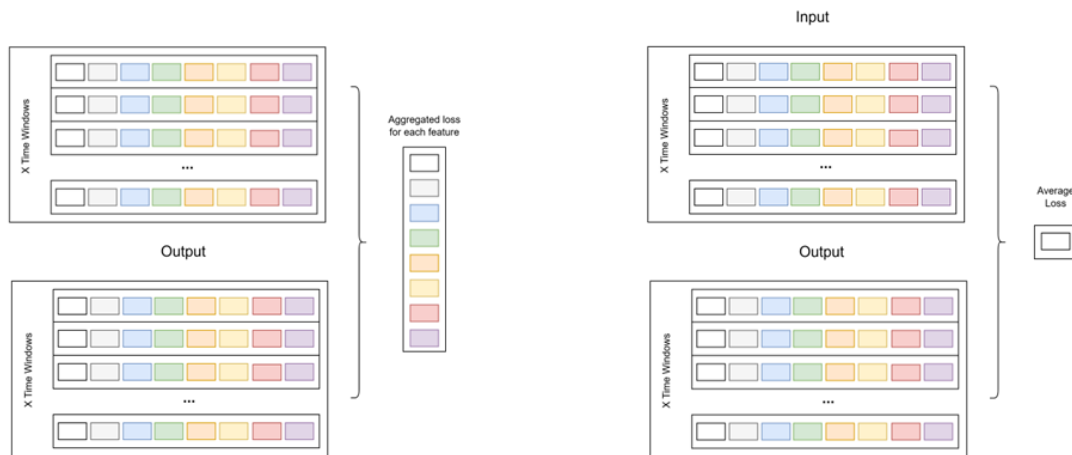


Figure 44 - Heuristic strategies for algorithm explanation

From this point, the XAI framework supported by the SHAP XAI algorithms creates an internal representation of the detection-algorithm workflow and information from the training dataset used. After the setup is completed and the XAI algorithms initialized, actual explanation is based on the following steps:

- Selection of a test instance for explanation;
- Perturbation of the instance to generate a dataset comprising similar instances;
- Assignment of weights to the similar instances based on their resemblance to the instance being explained;
- Training of a local, interpretable model using the weighted dataset;
- Using XAI-model weights to produce intelligible visualizations and dashboards on the impact of features for the decision making and DDoS detection.

With this approach, it possible to produce graphical visualizations on feature importance (see Figure 45), and general anomalous detection by feature in the timeseries input for a local instance of data in the detection windows of the timeseries with the timeseries algorithm.

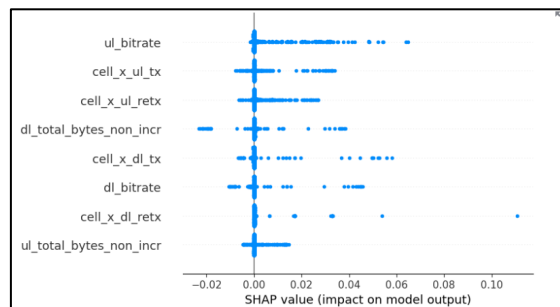


Figure 45 - Feature relevance for anomaly detection

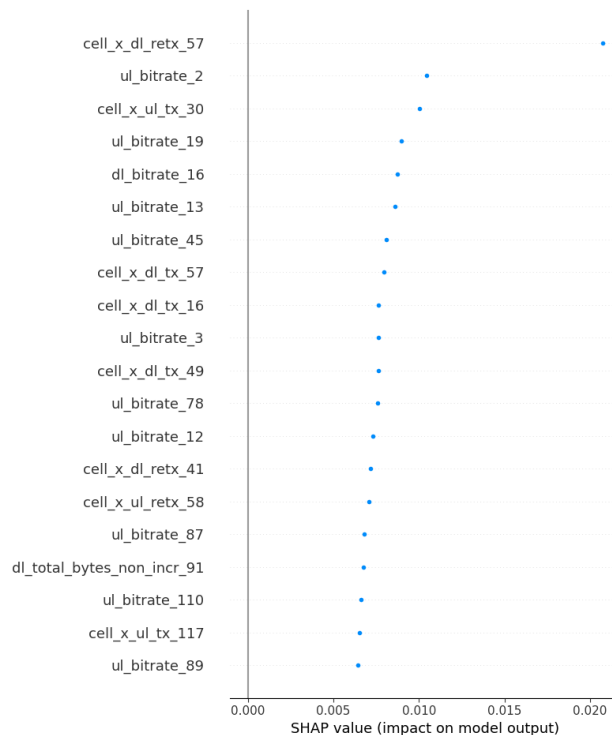


Figure 46 – Relevance of input features in timeseries for anomaly detection

In this context, we can affirm that the XAI framework is using:

- Instance-based counterfactual explainers by selecting the most similar examples from a dataset and assessing how a decision changes, or which input features modifications influence decision making in a XAI-weighted model;
- Meta-Explanation: usage of alternative XAI methods to create explanations based on the decision model. It might involve dedicated heuristics developed for a specific use case;
- Feature Importance: identification of the most important features for the explanation of a decision based on a model and instance of input;
- Examples: a strategy that aims to explain decisions based on similar examples from previous inputs.

The XAI framework is currently assessing opportunities for integration in workflow 2 and designing the integration of federation learning with local XAI explanations in a privacy-preserving format.

6.2.5.3 Hardware-accelerated analytics

PRIVATEER leverages Field-Programmable Gate Arrays (FPGAs) to accelerate the critical task of monitoring at runtime. Specifically, FPGAs are used to accelerate the ML-based anomaly-detection task described above. PRIVATEER provides an FPGA-LSTM (Long Short-Term Memory) application that targets only the inference stage of the LSTM-autoencoder model. This targeted acceleration ensures that the



computationally intensive inference operation is handled efficiently by the FPGA, leveraging the parallel-processing capabilities of the device, thus, providing significantly reduced latency and increased throughput for the monitoring analytics. While the FPGA implementation is not yet integrated into the core PRIVATEER platform, its integration is scheduled for Release B. This integration will create a more cohesive and robust solution, seamlessly combining the FPGA-accelerated LSTM inference with other necessary components within PRIVATEER.

LSTM Autoencoder Details: To achieve hardware acceleration, the initial step was to optimize the model architecture. Following the training procedures specified in the repository, we developed a more compact LSTM autoencoder model. The original autoencoder required a timestep window of 120 timesteps and included two LSTM layers each for the encoder and decoder. In contrast, the compact model operates with a timestep window of only 9 timesteps, featuring a single LSTM layer in the encoder and an LSTM layer coupled with a fully-connected layer in the decoder. Figure 47 presents the model summary of the implemented compact autoencoder.

```

=====
Layer (type:depth-idx)                Output Shape                Param #
=====
sm_LSTMAutoencoder                    [1, 9, 1]                  --
├─sm_Encoder: 1-1                      [1, 9, 9]                  --
│   └─LSTM: 2-1                         [1, 9, 9]                  432
├─sm_Decoder: 1-2                      [1, 9, 1]                  --
│   └─LSTM: 2-2                         [1, 9, 9]                  720
│       └─Linear: 2-3                    [1, 9, 1]                  10
=====
Total params: 1,162
Trainable params: 1,162
Non-trainable params: 0
Total mult-adds (M): 0.01

```

Figure 47 Overview of LSTM autoencoder layer architecture

We implemented the compact autoencoder model in an Alveo U280 FPGA. The encoder and decoder components are fully pipelined in the developed hardware architecture. Additionally, the design fully exploits the parallelism in the matrix-vector multiplications in the fully-connected and LSTM layers.

Preliminary Results: Figure 48 shows a terminal preview of the execution of the accelerated LSTM autoencoder application on the U280 FPGA device.

```

iccs@ai-at-edge-worker-01:~/lstm/v1$ ./lstm_app lstm.xclbin 1
# Starting Testbench
-----
Starting FPGA LSTM...
-----
Found Platform
Platform Name: Xilinx
INFO: Reading lstm.xclbin
Loading: 'lstm.xclbin'
Trying to program device[0]: xilinx_u280_xdma_201920_3
Device[0]: program successful!
Application compiled with NUM_CU = 1

input , 0.317466, -0.833132, 0.857289, 0.114122, -0.44833, 1.15243, -0.522378, 0.497939-----
-----
output , 0.284678, -0.915006, 1.05263, 0.00117624, -0.343249, 0.956469, -0.504089, 0.44386
# End of Testbench
-----
Performance Summary
-----
Dataset read time      :      0.00 ms
Device Initialization  :     262.32 ms
Buffer Allocation      :      0.50 ms
LSTM Computation       :      0.72 ms
Buffer Deallocation    :      0.57 ms

```

Figure 48 – Terminal preview of accelerated LSTM autoencoder execution on U280 FPGA

As shown Figure 48, the application workflow begins with loading the FPGA bitstream (lstm.xclbin) and the necessary runtime and device libraries. This initial step is crucial as it sets up the FPGA with the required configuration to perform the LSTM inference. The programming of the device involves transferring the bitstream into the FPGA, which effectively configures the hardware to perform the inference task using the LSTM kernel. Once the device is programmed, the input data is transferred through the Peripheral Component Interconnect Express (PCIe) interface into the Double Data Rate (DDR) memory of the FPGA (buffer allocation). The PCIe interface is used for high-speed data transfer between the host system and the FPGA, ensuring that the input data is quickly and efficiently moved to where it is needed. From the DDR memory, the data is transferred to the FPGA fabric, where the actual processing takes place. During execution, the LSTM kernel processes the input data (lstm computation), producing an output that is transferred back to the host system (buffer deallocation). As shown in Figure 47, the output closely matches the input, which is the expected behaviour of the LSTM autoencoder. Finally, the application provides a detailed performance summary of the end-to-end execution, with separate measurements for each critical step, offering insights into the efficiency and speed of each process. Overall, this setup provides a highly efficient and flexible system for accelerating LSTM inference, ready to be fully integrated into PRIVATEER release B.

The tables below highlight the resource usage, latency, and performance metrics of the hardware implementation. Table 4 demonstrates that our design comfortably fits within the U280 FPGA, with additional resources available for potential performance enhancements. Table 3 indicates a 15% reduction in latency for the hardware implementation compared to the compact model running on a CPU. With further optimizations, we anticipate achieving even greater speedups. Lastly, Table 5 confirms that our compact implementation meets the required accuracy standards.

Table 3 - Latency of accelerated LSTM autoencoder on U280 FPGA

Time over 1000 inferences (ms)	FPGA	CPU (PyTorch)
Buffer Allocation	217.72	-
LSTM Computation	441.72	-
Buffer Deallocation	175.67	-
Total	835.17	965.94

Table 4 - Resource utilization of accelerated LSTM autoencoder on U280 FPGA

Resources	Utilised	Utilisation %
BRAM_18K	182	4
DSP	5069	56
FF	649245	24
LUT	456010	34
URAM	0	0

Table 5 - Accuracy of accelerated LSTM autoencoder on U280 FPGA

Metric	FPGA
Accuracy	0.99
Precision	0.99
Recall	0.94
F1 Score	0.97
True Positives	94.18%
True Negatives	99.90%
False Positives	0.10%
False Negatives	5.82%

7 Workflow 3: Real-Time Mitigation of Man in the Middle Attacks

7.1 Attack models

The design and evolution of 5G towards B5G and 6G, will require the pervasive use of Network Function Virtualization (NFV) architectures and their co-existence with physical networking and with multiple different administrative domains with varying levels of security. In 6G this concept will be aligned to the idea of being a network of networks with support of dynamic domain reconfigurations that will make really challenging to clearly identify the network traffic paths. One of the main problems associated with this versatility will be the risk of MiTM (Man in The Middle) attack, where a malicious actor redirects, intercepts and may even alter the communication between two parties without them noticing. The most common MiTM attack models are the following:

- **Packet Sniffing:** captures data packets traveling through the network to extract sensitive data, shown in the following workflow (see Figure 49).
- **Session Hijacking:** the attacker steals the session token from the user and tries to impersonate with the aim of gaining access to unauthorized content.
- **SSL Stripping:** downgrade of the HTTPS connection to an HTTP connection by intercepting and modifying the initial request with the objective of capturing sensitive data.
- **DNS Spoofing:** corruption of DNS cache or responses, redirecting the network flow to a user non-desired malicious site.

7.2 Workflow 3 implementation

7.2.1 Short Description

For the real time mitigation of a Man in The Middle Attack, the Proof of Transit (PoT) will act as a Path Attestation method, discarding the packets that do not follow the trusted nodes.

7.2.2 Workflow

The flow begins with the deployment of an SDN (Software Defined Network) and configuration of the nodes (**steps 1 to 7, in Figure 49**). Following the paradigm of an SDN, the Controller, is in responsible for calculating and sharing different parameters to the nodes among the network through the P4Runtime API [9]. Once that all the nodes confirm the reception of the parameters, the node configuration procedure will conclude. Once the controller starts the collector, all the nodes will be capable of

executing and verifying the PoT logic as described on PRIVATEERs D4.1 [3] (**steps 9 to 28 in Figure 49**). Every time the Collector receives the PoT results and metrics, the Collector will act as the producer, sending the parameters to be consumed by the LoT Assessment manager through a Kafka Broker.

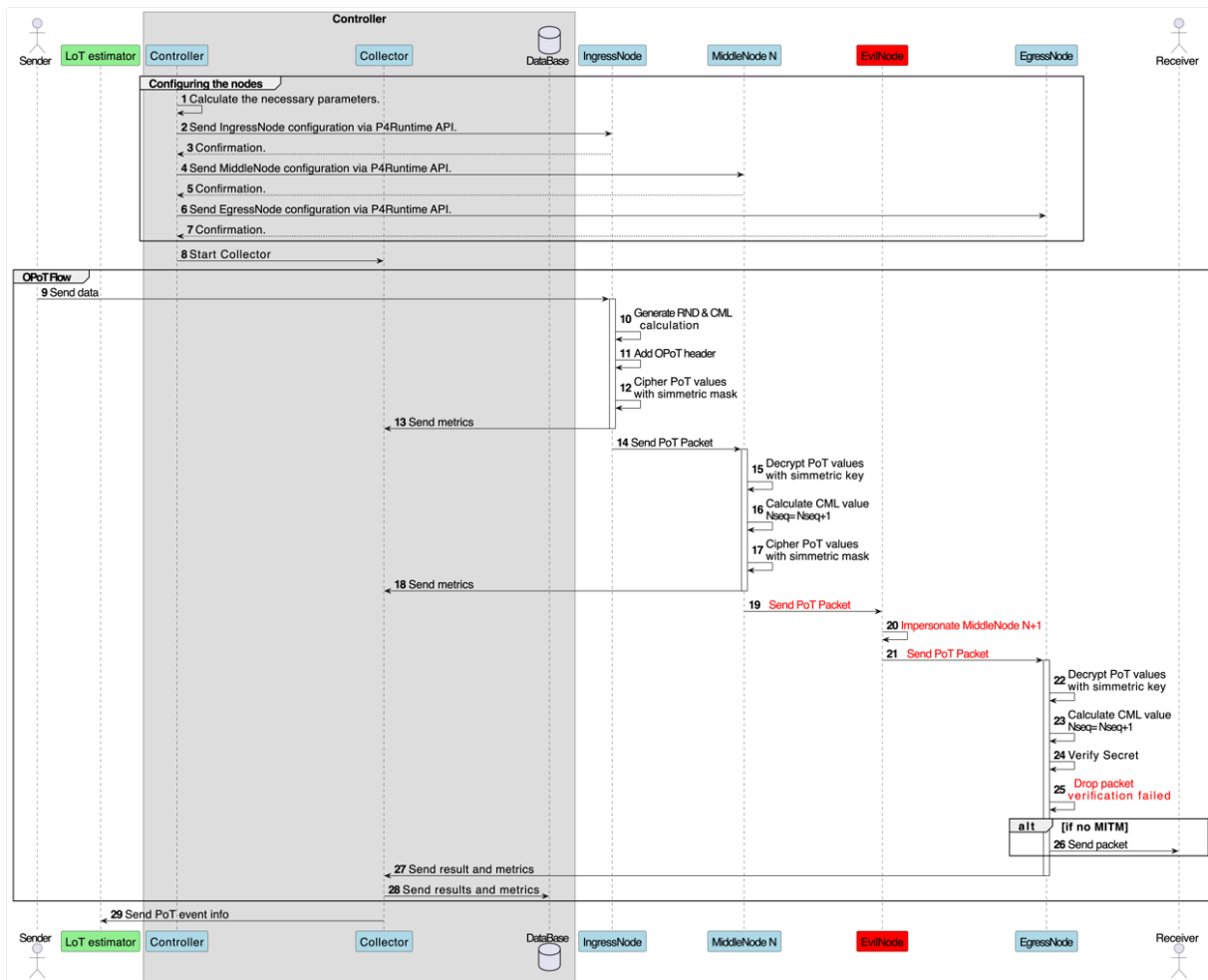


Figure 49 - MITM detection and mitigation sequence diagram

Figure 50 represents the PoT scenario with an evil node. The malicious node is intercepting the traffic by impersonating the middle node expected by the previous hop. Each component of the scenario is a Docker container. The process followed for testing the OPoT (Ordered Proof of Transit), where an additional mask is added, is the following one: host 1 (H1) generates traffic, IP packets, with host 2 (H2) destination. These packets go through P4 nodes, where metrics and actions related with the OPoT are implemented. The logic is done also by the Controller, responsible for the configuration of the nodes with OPoT process. Following the same steps mentioned in

PRIVATEER D4.1 Section 3.2.3 [3], the cumulative value CML mismatches with the expected value for the Egress node.

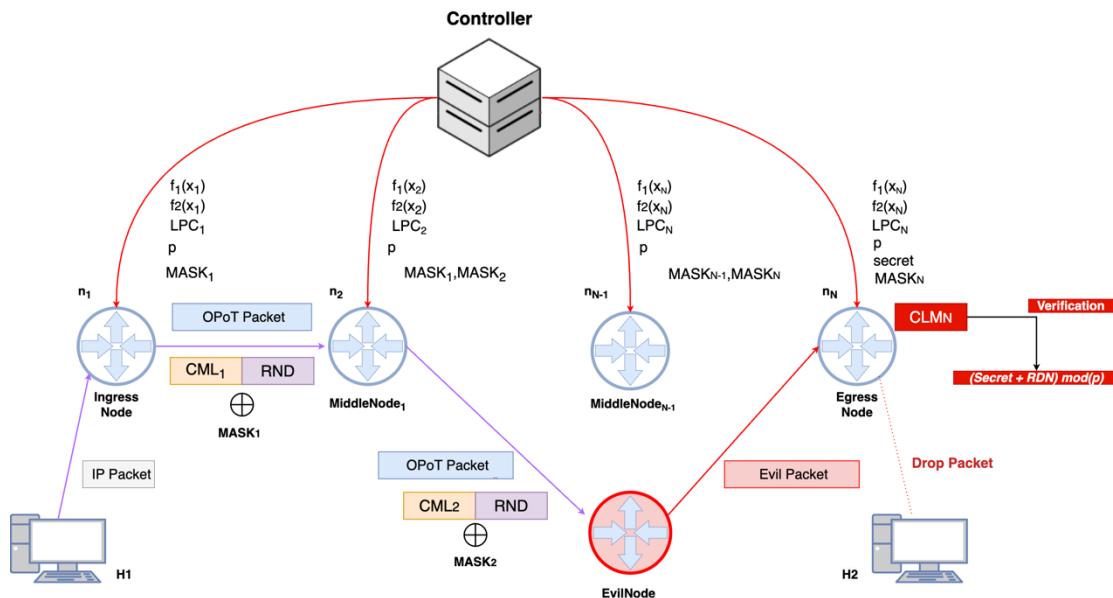


Figure 50: PoT MiTM scenario

For this case, $CML_N = (secret + RND) \pmod{p}$ is not fulfilled, the malicious node does not receive the necessary parameters from the controller for the PoT calculation. Consequently, when the evil node forwards the packet back to the following node on the established route, the OPoT parameters are not the expected ones. On the egress node, the verification process will fail, and the packet is discarded as a result, not arriving at the destination, H2 (see Figure 51).

```
ubuntu@PoT:~/PoT$ docker exec -it h1 /bin/sh
/ # ping 10.1.2.3 -c 20
PING 10.1.2.3 (10.1.2.3): 56 data bytes

--- 10.1.2.3 ping statistics ---
20 packets transmitted, 0 packets received, 100% packet loss
```

Figure 51: failed ping for the PoT scenario 2

Looking at the output of the data received by the controller, in the case of the scenario with the malicious node different parameters stand out (see Figure 52). On one hand, the middle nodes send information every five packets to the controller, indicating the correct operation (dropped=0). On the other, egress node sends information for each packet to the controller reporting that the PoT has failed between H1 and H2 (dropped=1). Another parameter that differs with the output of the scenario shown on PRIVATEERs D4.1 [3], is the Service Path Identifier, where the value is always equal to 55, relative to the ICMP echo request. Additionally, a new parameter has been generated for identification purposes, ServiceID, a UUID (Universally Unique



Identifier) that is generated using random numbers. In this case, as there are no responses from H2, packets related to ICMP echo reply do not appear, which is the case for the first scenario, with the Service Path Identifier equals to 23.

time	CML	Dropped	RND	Sequence Number	Service Index	Service Path Identifier	ServiceID	Switch IP
0	16	1	31	2	1	55	57310caf-bcc4-4008-82b8-6cfa6263bbcd	10.0.0.14
1001	51	1	9	3	1	55	57310caf-bcc4-4008-82b8-6cfa6263bbcd	10.0.0.14
2001	51	1	9	4	1	55	57310caf-bcc4-4008-82b8-6cfa6263bbcd	10.0.0.14
2999	17	0	6	5	3	55	57310caf-bcc4-4008-82b8-6cfa6263bbcd	10.0.0.11
3001	56	0	6	5	2	55	57310caf-bcc4-4008-82b8-6cfa6263bbcd	10.0.0.12
3003	4	1	6	5	1	55	57310caf-bcc4-4008-82b8-6cfa6263bbcd	10.0.0.14
4002	7	1	29	6	1	55	57310caf-bcc4-4008-82b8-6cfa6263bbcd	10.0.0.14
5002	60	1	11	7	1	55	57310caf-bcc4-4008-82b8-6cfa6263bbcd	10.0.0.14

Figure 52 - Output of the data received by the controller after failed PoT on the PoT scenario 2

7.2.3 APIs/Interfaces

For the integration with the LoT Assessment manager, a Kafka broker is used to save the metrics generated by the nodes to the collector, where the Collector is the producer for the Kafka broker and the LoT Assessment manager, the consumer. On Figure 53, the outcome of the “PoT-tests” Kafka topic is shown. The first lines, highlighted in green, show the scenario working correctly with no MiTM attack every 5 packets, where the field Dropped is equal to 0. The following lines, highlighted in pink, represent the outcome of the PoT facing the MiTM attack. The Dropped parameter is equal to 1 for the egress node, except for every 5 packets, highlighted in blue, where all the nodes inform of their current status. By having this information, it can be assumed that the nodes with the IPs 10.0.0.11 and 10.0.0.12 are the expected ones and in consequence, the attack has been done intercepting the packets for the node 10.0.0.13.

```

bash-5.1# ./kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic PoT-tests --from-beginning
{"measurement": "int telemetry", "time": 0, "fields": {"Switch IP": "10.0.0.11", "Service Path Identifier": 55, "Service Index": 3, "RND": 47,
"CML": 49, "Sequence Number": 5, "Dropped": 0, "ServiceID": "57310caf-bcc4-4008-82b8-6cfa6263bbcd"}}
{"measurement": "int telemetry", "time": 1, "fields": {"Switch IP": "10.0.0.12", "Service Path Identifier": 55, "Service Index": 2, "RND": 47,
"CML": 18, "Sequence Number": 5, "Dropped": 0, "ServiceID": "57310caf-bcc4-4008-82b8-6cfa6263bbcd"}}
{"measurement": "int telemetry", "time": 3, "fields": {"Switch IP": "10.0.0.13", "Service Path Identifier": 55, "Service Index": 1, "RND": 47,
"CML": 47, "Sequence Number": 5, "Dropped": 0, "ServiceID": "57310caf-bcc4-4008-82b8-6cfa6263bbcd"}}
{"measurement": "int telemetry", "time": 4, "fields": {"Switch IP": "10.0.0.14", "Service Path Identifier": 55, "Service Index": 0, "RND": 47,
"CML": 9, "Sequence Number": 5, "Dropped": 0, "ServiceID": "57310caf-bcc4-4008-82b8-6cfa6263bbcd"}}
{"measurement": "int telemetry", "time": 5, "fields": {"Switch IP": "10.0.0.14", "Service Path Identifier": 23, "Service Index": 3, "RND": 48,
"CML": 19, "Sequence Number": 5, "Dropped": 0, "ServiceID": "57310caf-bcc4-4008-82b8-6cfa6263bbcd"}}
{"measurement": "int telemetry", "time": 7, "fields": {"Switch IP": "10.0.0.13", "Service Path Identifier": 23, "Service Index": 2, "RND": 48,
"CML": 27, "Sequence Number": 5, "Dropped": 0, "ServiceID": "57310caf-bcc4-4008-82b8-6cfa6263bbcd"}}
{"measurement": "int telemetry", "time": 8, "fields": {"Switch IP": "10.0.0.12", "Service Path Identifier": 23, "Service Index": 1, "RND": 48,
"CML": 32, "Sequence Number": 5, "Dropped": 0, "ServiceID": "57310caf-bcc4-4008-82b8-6cfa6263bbcd"}}
{"measurement": "int telemetry", "time": 9, "fields": {"Switch IP": "10.0.0.11", "Service Path Identifier": 23, "Service Index": 0, "RND": 48,
"CML": 10, "Sequence Number": 5, "Dropped": 0, "ServiceID": "57310caf-bcc4-4008-82b8-6cfa6263bbcd"}}
{"measurement": "int telemetry", "time": 0, "fields": {"Switch IP": "10.0.0.14", "Service Path Identifier": 55, "Service Index": 1, "RND": 31,
"CML": 16, "Sequence Number": 2, "Dropped": 1, "ServiceID": "57310caf-bcc4-4008-82b8-6cfa6263bbcd"}}
{"measurement": "int telemetry", "time": 1001, "fields": {"Switch IP": "10.0.0.14", "Service Path Identifier": 55, "Service Index": 1, "RND": 9,
"CML": 51, "Sequence Number": 3, "Dropped": 1, "ServiceID": "57310caf-bcc4-4008-82b8-6cfa6263bbcd"}}
{"measurement": "int telemetry", "time": 2001, "fields": {"Switch IP": "10.0.0.14", "Service Path Identifier": 55, "Service Index": 1, "RND": 9,
"CML": 51, "Sequence Number": 4, "Dropped": 1, "ServiceID": "57310caf-bcc4-4008-82b8-6cfa6263bbcd"}}
{"measurement": "int telemetry", "time": 2999, "fields": {"Switch IP": "10.0.0.11", "Service Path Identifier": 55, "Service Index": 3, "RND": 6,
"CML": 17, "Sequence Number": 5, "Dropped": 0, "ServiceID": "57310caf-bcc4-4008-82b8-6cfa6263bbcd"}}
{"measurement": "int telemetry", "time": 3001, "fields": {"Switch IP": "10.0.0.12", "Service Path Identifier": 55, "Service Index": 2, "RND": 6,
"CML": 56, "Sequence Number": 5, "Dropped": 0, "ServiceID": "57310caf-bcc4-4008-82b8-6cfa6263bbcd"}}
{"measurement": "int telemetry", "time": 3003, "fields": {"Switch IP": "10.0.0.14", "Service Path Identifier": 55, "Service Index": 1, "RND": 6,
"CML": 4, "Sequence Number": 5, "Dropped": 1, "ServiceID": "57310caf-bcc4-4008-82b8-6cfa6263bbcd"}}
{"measurement": "int telemetry", "time": 4002, "fields": {"Switch IP": "10.0.0.14", "Service Path Identifier": 55, "Service Index": 1, "RND": 2,
"CML": 7, "Sequence Number": 6, "Dropped": 1, "ServiceID": "57310caf-bcc4-4008-82b8-6cfa6263bbcd"}}
{"measurement": "int telemetry", "time": 5002, "fields": {"Switch IP": "10.0.0.14", "Service Path Identifier": 55, "Service Index": 1, "RND": 1,
"CML": 60, "Sequence Number": 7, "Dropped": 1, "ServiceID": "57310caf-bcc4-4008-82b8-6cfa6263bbcd"}}

```

Figure 53: Kafka PoT-tests topic outcome

7.2.4 Preliminary Results

The process described below outlines the steps involved in processing messages and evaluating the PoT and the LoT (Level of Trust) values using fuzzy logic. This integration allows for a robust and scalable solution for real-time data processing.

Experiments:

- 1 **Ping Sent Between Nodes:** A ping is sent between nodes to initiate the PoT execution (Figure 54).

```
ping 10.1.2.3 -c 10
```

Figure 54 - Ping address

Reading and Processing Messages: Messages produced by PoT are read, and the input is processed (Figure 55).

```
{
  "measurement": "int_telemetry",
  "time": 4143541,
  "fields": {
    "Switch IP": "10.0.0.13",
    "Service Path Identifier": 55,
    "Service Index": 1,
    "RND": 41,
    "CML": 0,
    "Sequence Number": 15,
    "Dropped": 0,
    "ServiceID": "57310caf-bcc4-4008-82b8-6cfa6263bbcd"
  }
}

{
  "measurement": "int_telemetry",
  "time": 4143542,
  "fields": {
    "Switch IP": "10.0.0.14",
    "Service Path Identifier": 55,
    "Service Index": 0,
    "RND": 41,
    "CML": 1,
    "Sequence Number": 15,
    "Dropped": 0,
    "ServiceID": "57310caf-bcc4-4008-82b8-6cfa6263bbcd"
  }
}
```

Figure 55 - PoT in JSON format

Task Creation: A new task is created based on the processed messages (Figure 56) and insert to a new row in the MongoDB Database (Figure 57).



```
[data-collector] [ntainer#1-0-C-1] o.v.d.s.impl.AggregationServiceImpl : Lot-Fuzzy Assessment started by pot event
[data-collector] [ntainer#1-0-C-1] o.v.d.s.impl.AggregationServiceImpl : POT___DATA:: {time=4971814, fields={Switch IP=10.0.0.12, Service Path Identifier=23, Service Index=
[data-collector] [ntainer#1-0-C-1] o.v.d.s.impl.AggregationServiceImpl : TaskInfo pot -> org.ucm.datacollector.model.TaskInfo@79728319
[data-collector] [ntainer#1-0-C-1] o.v.d.s.impl.AggregationServiceImpl : Lot-Fuzzy Assessment started by pot event
[data-collector] [ntainer#1-0-C-1] o.v.d.s.impl.AggregationServiceImpl : POT___DATA:: {time=4971810, fields={Switch IP=10.0.0.11, Service Path Identifier=23, Service Index=
[data-collector] [ntainer#1-0-C-1] o.v.d.s.impl.AggregationServiceImpl : TaskInfo pot -> org.ucm.datacollector.model.TaskInfo@80529031f
[data-collector] [ntainer#1-0-C-1] o.v.d.s.impl.AggregationServiceImpl : THE FUZZY TASK WILL BE STARTED
[data-collector] [ntainer#1-0-C-1] o.v.s.r.f.client.ExchangeFunctions : [20c27094] HTTP GET http://localhost:8082/dimension/1
[data-collector] [or-http-spoll-2] o.s.w.r.f.client.ExchangeFunctions : [20c27094] [41f87418-1] Response 200 OK
[data-collector] [or-http-spoll-2] org.springframework.web.HttpLogging : [20c27094] [41f87418-1] Decoded [org.ucm.datacollector.model.D10282f437f]
[data-collector] [ntainer#1-0-C-1] o.v.s.w.r.f.client.ExchangeFunctions : [79019178] HTTP GET http://localhost:8082/dimension/2
[data-collector] [or-http-spoll-3] o.s.w.r.f.client.ExchangeFunctions : [79019178] [9c29e8e-1] Response 200 OK
[data-collector] [or-http-spoll-3] org.springframework.web.HttpLogging : [79019178] [9c29e8e-1] Decoded [org.ucm.datacollector.model.D20273e1a77]
[data-collector] [ntainer#1-0-C-1] o.v.s.w.r.f.client.ExchangeFunctions : [13fcefd2] HTTP GET http://localhost:8082/dimension/3
[data-collector] [or-http-spoll-2] o.s.w.r.f.client.ExchangeFunctions : [13fcefd2] [41f87418-2] Response 200 OK
```

Figure 56 - Task creation process

```
_id: ObjectId('6685643ed9cda05bb7666559')
driven_type: "WARN"
created_by: "POT"
event_date: 2024-07-03T14:46:22.574+00:00
task_date: 2024-07-03T14:46:22.675+00:00
status: 1
_class: "org.ucm.datacollector.model.lot.entity.Task"
```

Figure 57 - Task table, specific Row related to PoT

Fuzzy Logic Evaluation (Stage 1): The first step of fuzzy logic is the evaluation of the inputs that we receive based on Time-driven or Event-driven logic (Figure 58), the result saved on DB (Figure 59).

```
{"_id": "66880c35ff5665075nc10b7c", "taskId": "66880c359cf44f3aaeb67560", "recordDate": [2024, 7, 5, 15, 7, 33, 976567083], "d1": "49.32653668745014", "d2": "50.00000000000004", "d3": "50.00000000000056", "d4": "9923", "d5": "83.29999999999923", "d6": null}
```

Figure 58 - Result of the first step of LoT Assessment

```
_id: ObjectId('66856454e9983477ebe52560')
task_id: "6685643ed9cda05bb7666559"
D1: 54.666800886740404
D2: 39.526917592160885
D3: 22.779812925794495
D4: 50.00000000000004
D5: 0
record_date: 2024-07-03T14:46:44.129+00:00
_class: "org.ucm.lotfuzzy.model.lot.entity.DimensionsAssessment"
```

Figure 59 - Dimensions table

LoT Evaluation: The LoT evaluation is completed, providing an overall assessment (Figure 60), eventually saved on DB (Figure 61).

```
{"_id": "66880c35ff5665075nc10b7c", "taskId": "66880c359cf44f3aaeb67560", "recordDate": [2024, 7, 5, 15, 7, 33, 976567083], "d1": "49.32653668745014", "d2": "50.00000000000004", "d3": "50.00000000000056", "d4": "9923", "d5": "83.29999999999923", "d6": null}
```

Figure 60 - Final results of the LoT assessment shown in different ways

```
_id: ObjectId('66856454e9983477e52560')
task_id: "6685643ed9cda95bb7666559"
D1: 54.666800886740404
D2: 39.52691759216885
D3: 22.779812925794495
D4: 50.0000000000004
D5: 0
record_date: 2024-07-03T14:46:44.129+00:00
_class: "org.ucm.lotfuzzy.model.lot.entity.DimensionsAssessment"
```

Figure 61 - Final_Trust Result



8 Conclusion & next steps

The demonstration of the Release A of the PRIVATEER Framework shows the significant progress made toward addressing the project's objectives. The three workflows serve as verification and evidence of the interoperability between components and comprise a stepping stone toward the project's Release B.

These workflows also showcase the progress and achievements of the initial integration activities between components of the different Work Packages. Each workflow was chosen to implement specific parts of the originally identified PRIVATEER Use Cases originally identified in D2.2. This mapping is described in Annex B in detail. Future work will focus on expanding the functionalities and integration capabilities, leading up to the second -and final- release scheduled for M36 of the project.

References

- [1] *PRIVATEER project "Deliverable D2.2 - Use cases, requirements and design report"*.
- [2] *PRIVATEER project "Deliverable D3.1 - Decentralised Robust Security Analytics Enablers Rel. A"*.
- [3] *PRIVATEER project "Deliverable 4.1 - Privacy-aware slicing and orchestration enablers – Rel. A"*.
- [4] *PRIVATEER project "Deliverable D5.1 - Distributed attestation, identity and threat sharing enablers – Rel. A. - ToC and next steps"*.
- [5] "A Technical Analysis of Confidential Computing," Confidential Computing Consortium, 2022.
- [6] S. Kapil, K. Veronika, S. Jon, L. David, O. Seosamh, C. Darragh and K. Lorek, *Intel® Software Guard Extensions (Intel® SGX) – Key Management Reference Application (KMRA) on the 3rd and 4th Gen Intel® Xeon® Scalable Processors*, 2022.
- [7] ETSI, *"Network Functions Virtualisation (NFV): Trust: Report on Attestation Technologies and Practices for Secure Deployments*, 2017.
- [9] "P4 Runtime Specification," [Online]. Available: <https://p4.org/p4-spec/p4runtime/main/P4Runtime-Spec.html>.
- [10] "Whitepaper Reference Architecture for Confidential Computing on SKT 5G MEC," Intel, 2021.



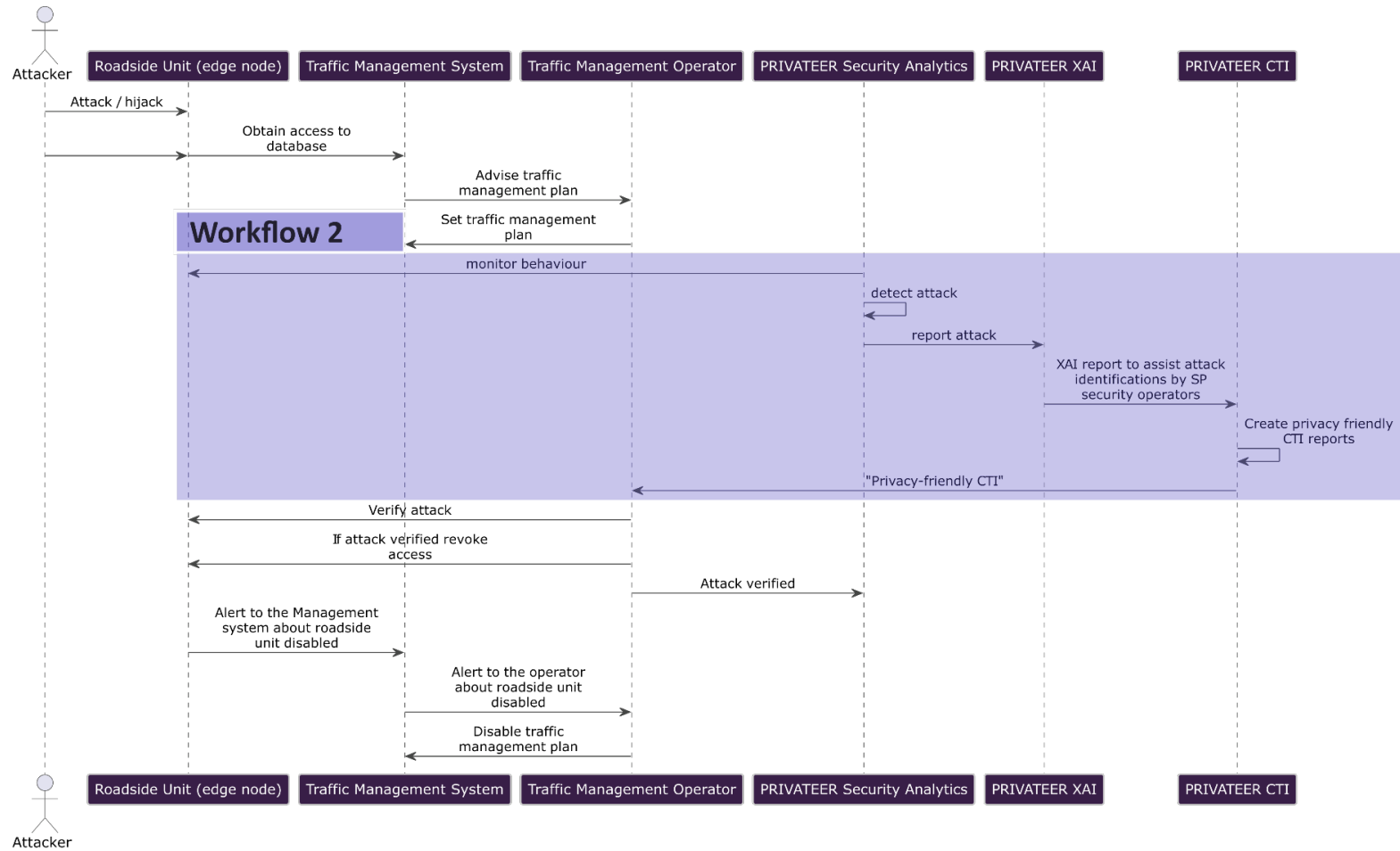
Annex A: Mapping of Workflows to PRIVATEER Use Cases

The workflows presented in this deliverable constitute an initial step towards the implementation of the PRIVATEER Use Cases, described in D2.2. The Table 6 summarizes the mapping of the Workflows to the Use Cases as of Release A of the PRIVATEER Demonstrator. Below, we also present in detail the parts of the UCs sequence diagrams that are currently covered by the Workflows. The remaining integrations for implementing the UCs sequence diagrams and scenarios detailed in D2.2 will take place and be fully demonstrated in Release B.

Table 6 Workflows to PRIVATEER Use Cases Mapping

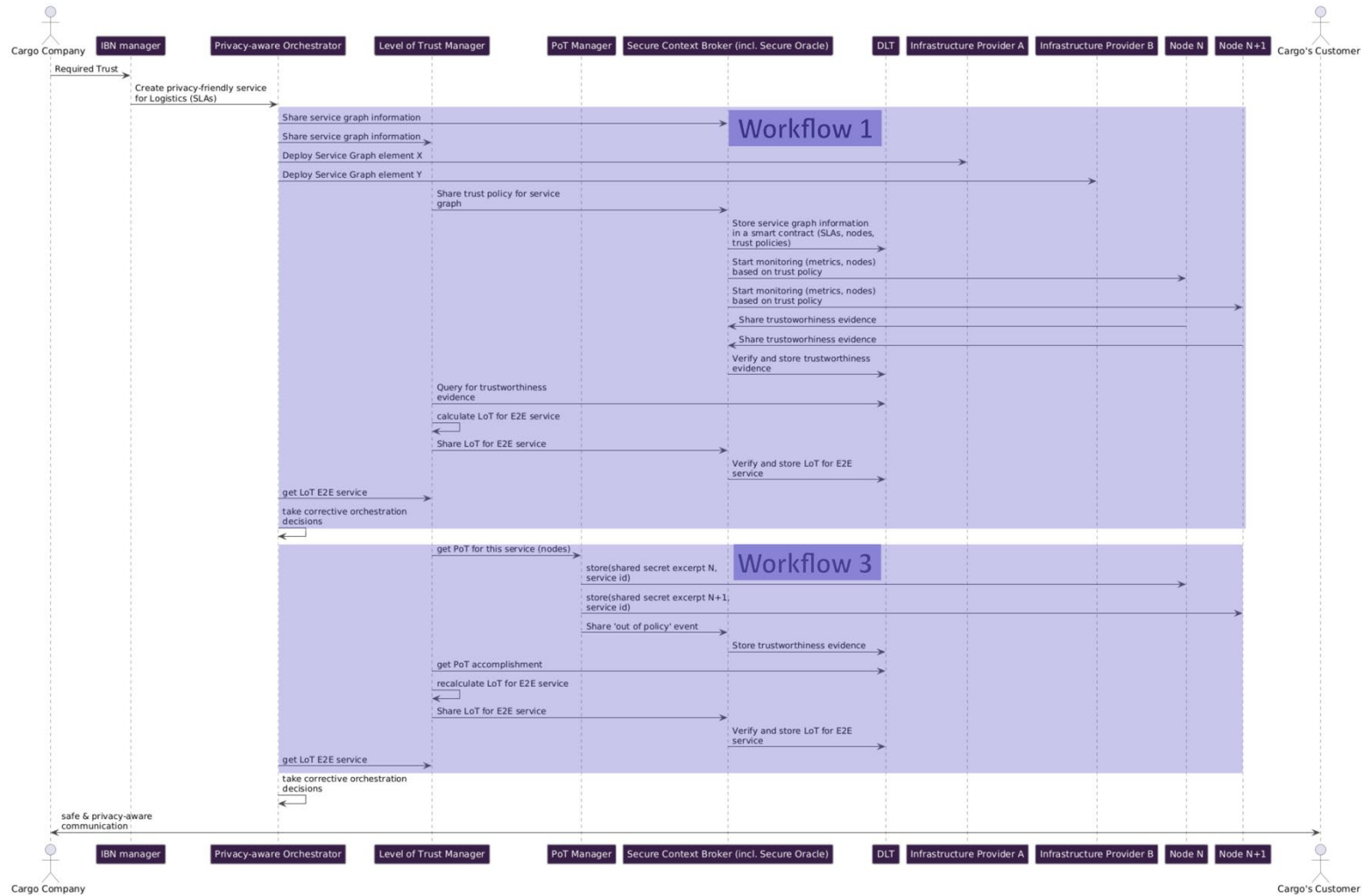
Use Case	Related Workflows
UC1 – Edge Service Compromise	W2
UC2 - Privacy-friendly security service orchestration for logistics	W1, W3
UC3 - Verification of Mass Transportation application (phase 2)	W1
UC4 - Onboarding of neutral host edge network	W2
UC5 - Multi-domain infrastructure verification and PoT	W3

UC1 & Workflow 2 Mapping



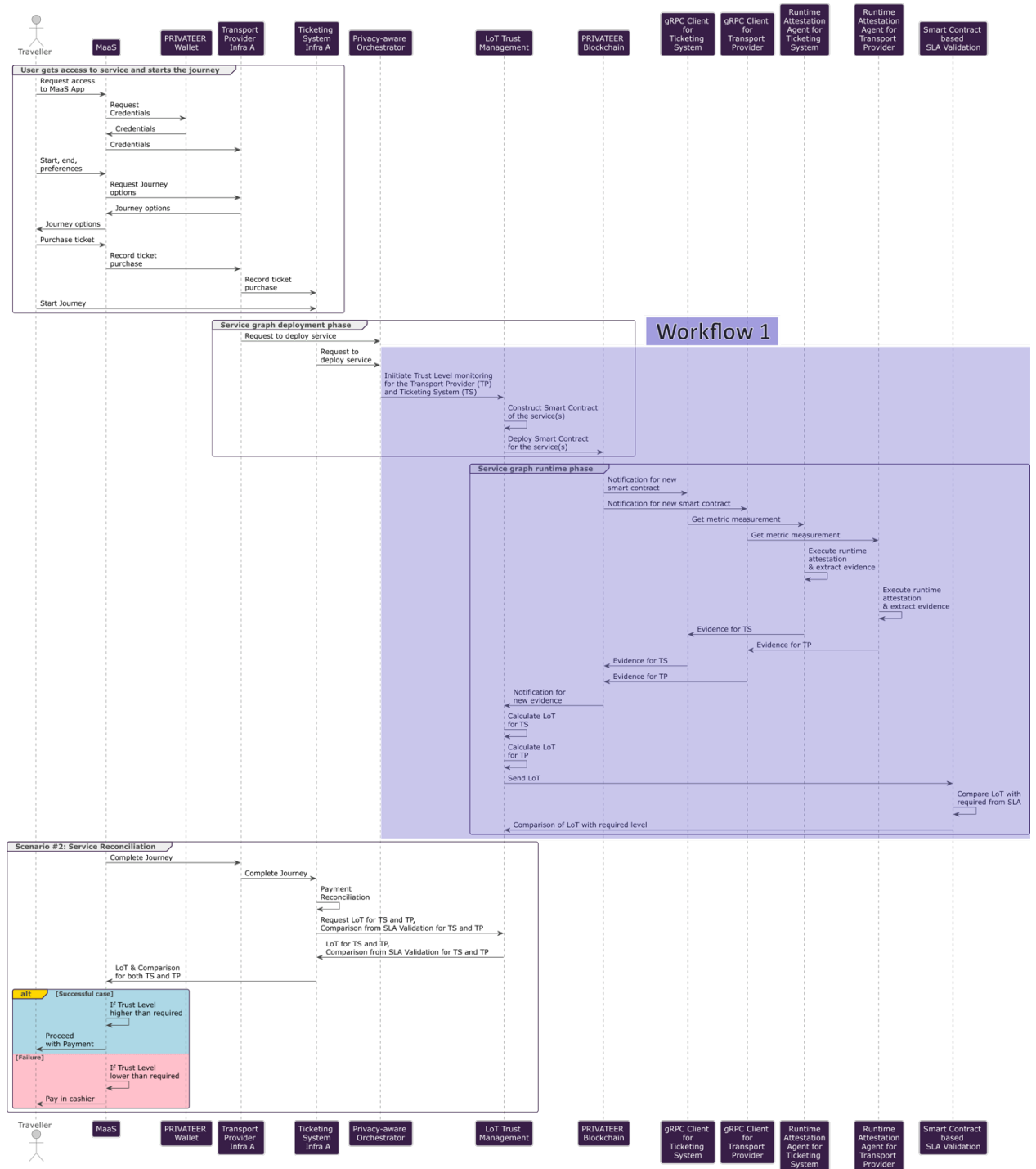


UC2 to Workflows 1 and 3 Mapping



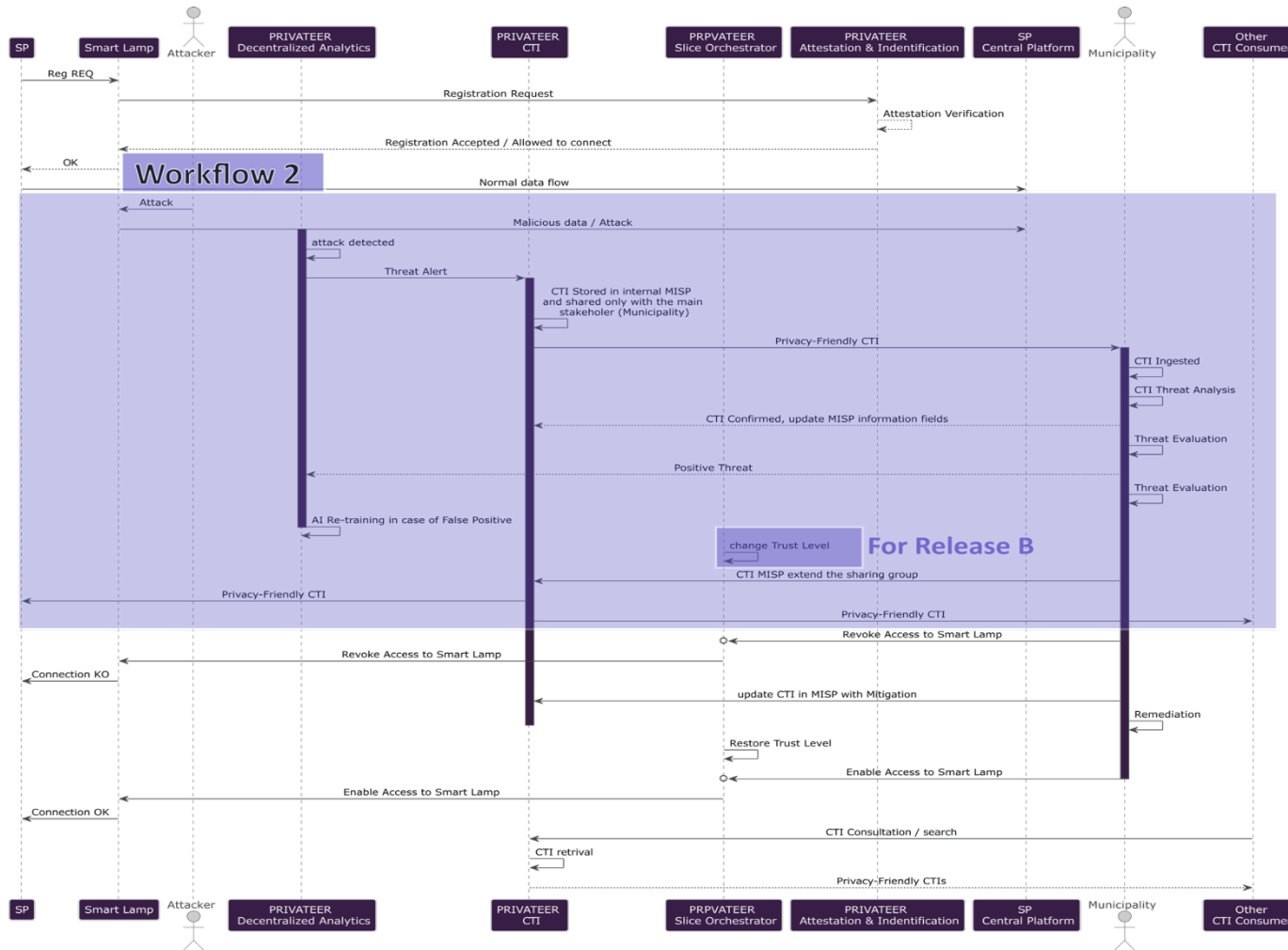


UC3 to Workflow 1 Mapping

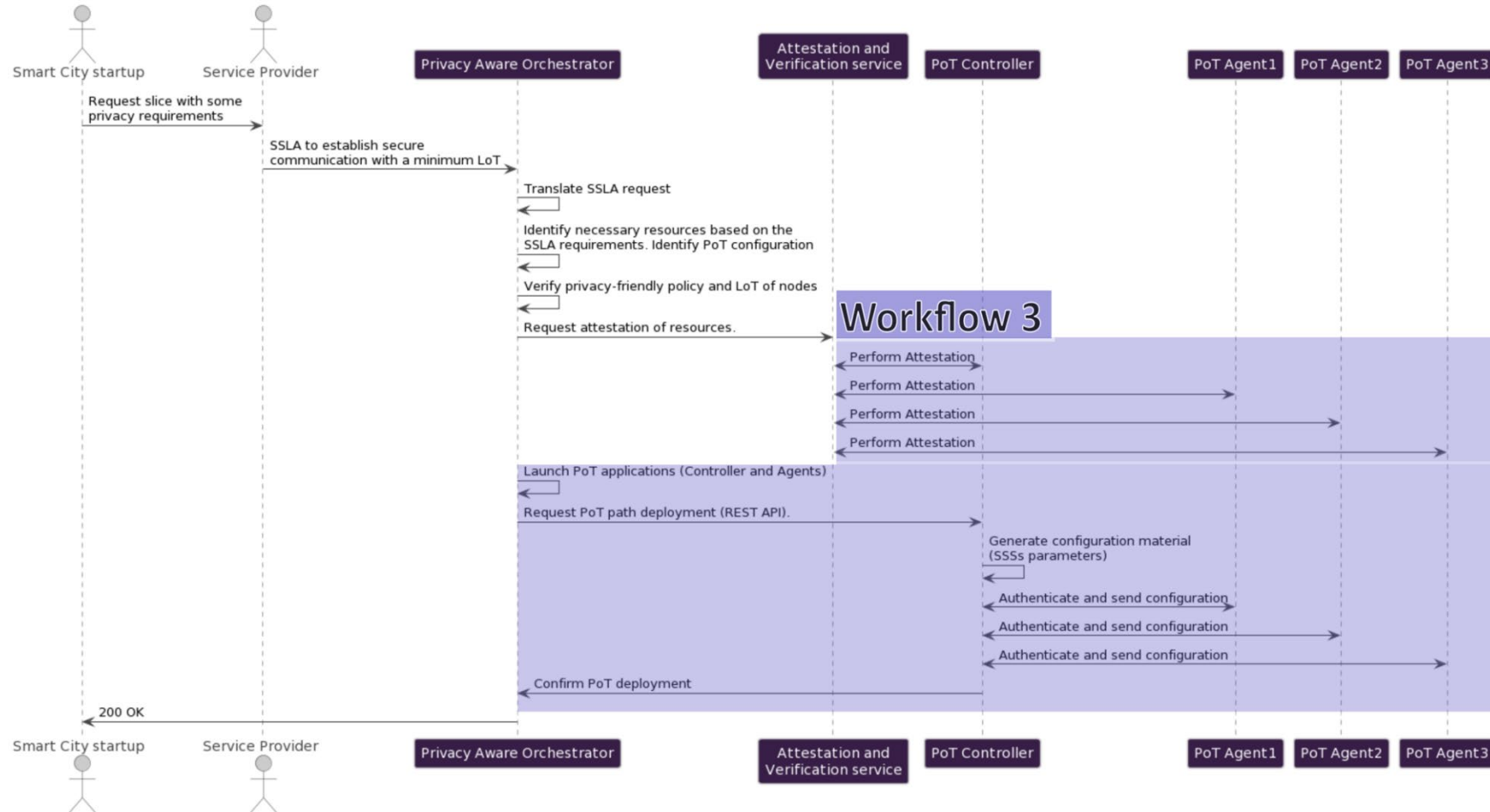




UC4 to Workflow 2 Mapping



UC5 to Workflow 3 Mapping





Consortium



Space Hellas
www.space.gr



DEMOKRITOS

NCSR Demokritos
www.demokritos.gr



Telefónica

Telefonica I&D
www.telefonica.com



RHEA SYSTEM SA
www.rheagroup.com



INESC TEC
www.inesctec.pt



Infili Technologies PC
www.infili.com



UBITECH LTD
www.ubitech.eu



IQUADRAT R&D
www.ucm.es



ICCS
www.iccs.gr



**FORSVARETS
FORSKNINGSINSTITUTT**
www.ffi.no



**UNIVERSIDAD
COMPLUTENSE DE MADRID**
www.ucm.es



**INSTITUTO POLITÉCNICO
DO PORTO**
www.ipp.pt



ERTICO ITS EUROPE
www.ertico.com

Contact Us

privateer-contact@spacemaillist.eu



PRIVATEER has received funding from the Smart Networks and Services Joint Undertaking (SNS JU) under the European Union's Horizon Europe research and innovation programme under Grant Agreement No. 101096110